GE
Intelligent Platforms

***State Logic Products***

# *Series 90 -70*
# *State Logic Control System*

## *User's Manual*

*GFK-1006A* *March* 2010

## *Warnings, Cautions, and Notes as Used in this Publication*

**Warning**

**Warning notices are used in this publication to emphasize that hazardous voltages, currents, temperatures, or other conditions that could cause personal injury exist in this equipment or may be associated with its use.**

**In situations where inattention could cause either personal injury or damage to equipment, a Warning notice is used.**

**Caution**

**Caution notices are used where equipment might be damaged if care is not taken.**

**Note:** Notes merely call attention to information that is especially significant to understanding and operating the equipment.

This document is based on information available at the time of its publication. While efforts have been made to be accurate, the information contained herein does not purport to cover all details or variations in hardware or software, nor to provide for every possible contingency in connection with installation, operation, or maintenance. Features may be described herein which are not present in all hardware and software systems. GE Intelligent Platforms assumes no obligation of notice to holders of this document with respect to changes subsequently made.

GE Intelligent Platforms makes no representation or warranty, expressed, implied, or statutory with respect to, and assumes no responsibility for the accuracy, completeness, sufficiency, or usefulness of the information contained herein. No warranties of merchantability or fitness for purpose shall apply.

* indicates a trademark of GE Intelligent Platforms, Inc. and/or its affiliates. All other trademarks are the property of their respective owners.

**©Copyright 2010 GE Intelligent Platforms, Inc.**
**All Rights Reserved**

# Contact Information

If you purchased this product through an Authorized Channel Partner, please contact the seller directly.

*General Contact Information*

| | |
|---|---|
| Online technical support and GlobalCare | http://www.ge-ip.com/support |
| Additional information | http://www.ge-ip.com/ |
| Solution Provider | solutionprovider.ip@ge.com |

*Technical Support*

If you have technical problems that cannot be resolved with the information in this guide, please contact us by telephone or email, or on the web at www.ge-ip.com/support

*Americas*

| | |
|---|---|
| Online Technical Support | www.ge-ip.com/support |
| Phone | 1-800-433-2682 |
| International Americas Direct Dial | 1-780-420-2010 (if toll free 800 option is unavailable) |
| Technical Support Email | support.ip@ge.com |
| Customer Care Email | customercare.ip@ge.com |
| Primary language of support | English |

*Europe, the Middle East, and Africa*

| | |
|---|---|
| Online Technical Support | www.ge-ip.com/support |
| Phone | +800-1-433-2682 |
| EMEA Direct Dial | +352-26-722-780 (if toll free 800 option is unavailable or if dialing from a mobile telephone) |
| Technical Support Email | support.emea.ip@ge.com |
| Customer Care Email | customercare.emea.ip@ge.com |
| Primary languages of support | English, French, German, Italian, Czech, Spanish |

*Asia Pacific*

| | |
|---|---|
| Online Technical Support | www.ge-ip.com/support |
| Phone | +86-400-820-8208 |
| | +86-21-3217-4826 (India, Indonesia, and Pakistan) |
| Technical Support Email | support.cn.ip@ge.com (China) |
| | support.jp.ip@ge.com (Japan) |
| | support.in.ip@ge.com (remaining Asia customers) |
| Customer Care Email | customercare.apo.ip@ge.com |
| | customercare.cn.ip@ge.com (China) |

## Content of this Manual

**Chapter 1. Preliminaries**

**Chapter 2. Setting Up the 90-70 State Logic Control System**

**Chapter 3. State Logic® Control Theory**

**Chapter 4. Creating a State Logic Control Program**

**Chapter 5. ECLiPS Programming Features**

**Chapter 6. Program Instructions**

**Chapter 7. Perform Functions**

**Chapter 8. PID Loops**

**Chapter 9. On-Line Features**

**Chapter 10. Serieal Communications Module**

**Appendix A. Key Functions**

**Appendix B. Language Structure Summary**

**Appendix C. References to the Genius PowerTrac Block**

**Appendix D. References to Genius High Speed Counter Block**

**Appendix E. Errors**

**Appendix F. Standard Predefined Keywords**

**Appendix G. Integrating Ladder Logic and/or 'C' Programming**

**Appendix H. ECLiPS Specifications**

**INDEX**

## We Welcome Your Comments and Suggestions

At GE Intelligent Platforms, we strive to produce quality technical documentation. After you have used this manual, please take a few moments to complete and return the Reader's Comment Card located on the next page.

® State Logic is a registered trademark of Adatek, Inc.

**Chapter 1 Preliminaries ..... 1-1**

Items Included with ECLiPS ..... 1-1
Product Registration ..... 1-2
Getting Answers to Your State Logic Questions ..... 1-2
How to Use this Manual ..... 1-3
Brief Description of the Manual Chapters ..... 1-3
Manual Organization ..... 1-5

**Chapter 2 Setting Up the 90-70 State Logic Control System ..... 2-1**

System Overview (Architecture) ..... 2-2
Connecting ECLiPS, OnTOP, and Logicmaster 90 to the State Engine ..... 2-3
Setting Up ECLiPS ..... 2-3
Hardware Requirements ..... 2-3
ECLiPS Installation ..... 2-4
Hardware Key ..... 2-4
Configuring ECLiPS Memory Usage ..... 2-4
Loading Logicmaster Hardware Configuration ..... 2-5
State Engine Configuration Options ..... 2-6
Security Setup ..... 2-6
ECLiPS Security ..... 2-6
OnTOP Security System ..... 2-7
Setting Up OnTOP ..... 2-7
Installation ..... 2-7
Using ECLiPS Program Files ..... 2-8
Setting Up Logicmaster 90 ..... 2-8
Setting Up the State Engine ..... 2-8
Loading the State Engine Operating System ..... 2-8
Configuring the State Engine Hardware System ..... 2-9
Communicating with the State Engine ..... 2-10
Communicating with the Programmer Software Packages ..... 2-10
Serial Communication ..... 2-10
EtherNet Communication ..... 2-11

**Chapter 3 State Logic Control Theory ..... 3-1**

State Logic Control Theory ..... 3-2
The Concept of Finite States ..... 3-2
What Makes State Control Logic Different ..... 3-4
A Collection of Tasks is a State Logic Program ..... 3-4
Developing State Logic Programs with ECLiPS ..... 3-5
Tasks - Sequences of States ..... 3-6
States - The Building Blocks of a Task ..... 3-6
Statements - The Command Set for State Descriptions ..... 3-7
Scan Overview ..... 3-9
Interaction Between Tasks ..... 3-9
Creating Process Diagnostics ..... 3-11

**Chapter 4 Creating a State Logic Control Program ..... 4-1**

Outline the Application ..... 4-2
Identify the Tasks ..... 4-2
Identify The States ..... 4-2
Identify the Statements ..... 4-3
Writing The Program ..... 4-4
Using English Names in the ECLiPS Program ..... 4-4
Statement Structures ..... 4-6
Constructing Statements ..... 4-6
Using Keywords, Synonyms and Filler Words ..... 4-7
Using Variables ..... 4-7
Program Scan ..... 4-9
Hints for Creating ECLiPS Programs ..... 4-12
Outputs are OFF by Default ..... 4-12
Task Design ..... 4-14
Write Term Considerations ..... 4-14
Calculations and a Scanning Operating System ..... 4-15
Read Term Considerations ..... 4-15
Timer Considerations ..... 4-16
Documentation Hints ..... 4-17
Programming Conventions ..... 4-18
Scan Time Considerations ..... 4-18

**Chapter 5 ECLiPS Programming Features ..... 5-1**

State Logic Word Processor ..... 5-2
Creating Program Text - Overview ..... 5-2
Text Functions - Block Functions ..... 5-3
Find Functions - Search and Replace ..... 5-3
Project Management ..... 5-4
File Management ..... 5-4
Documentation - Print Function ..... 5-5
Error Checks, Translate, and Download Projects ..... 5-5
Task Groups ..... 5-6
File Types Created by ECLiPS ..... 5-6
Naming Variables ..... 5-7
Define Current Word - Search for Undefined Words ..... 5-7
List - Variable Type ..... 5-7
The System Configuration Option ..... 5-7
On-Line Program Changes ..... 5-8
Restrictions ..... 5-8
Hints for Using ECLiPS Features ..... 5-8
How to Use the ECLiPS Menus ..... 5-8
Using ECLiPS Hot Keys ..... 5-8
ECLiPS Word Processing Functions ..... 5-9
How to Use ECLiPS Lists ..... 5-9

**Chapter 6 Program Instructions ..... 6-1**

Program Structure ..... 6-2
Task Groups ..... 6-2
Tasks ..... 6-3
State ..... 6-3
Statements ..... 6-3
Expressions ..... 6-3
Terms ..... 6-4
Words ..... 6-4
Functional Terms ..... 6-5
Turning ON Discretes (Actuate Term) ..... 6-5
Assigning Analog and Variable Values ..... 6-5
Changing Active States Term ..... 6-7
Sending Serial Data (Write Term) ..... 6-8
PID Loops Control Terms (Start_PID, Stop_PID) ..... 6-9
Change Serial Port Configuration Term ..... 6-10
Perform Function Term ..... 6-10
Conditional Terms ..... 6-10
Digital Conditional Term ..... 6-11
Timer Conditional Term ..... 6-11
Relational Conditional Term ..... 6-12
Current State Conditional Term ..... 6-12
Complex Conditionals ..... 6-13
Character Input Conditional Term ..... 6-13
Mathematical Calculations ..... 6-14
Operator Precedence ..... 6-14
Variables ..... 6-14
ASCII Variables ..... 6-15
Numeric Variables ..... 6-15
Numeric Data Types ..... 6-16
Grammatical Rules ..... 6-16
Filler Words ..... 6-16

**Chapter 7 Perform Funtions ..... 7-1**

Table Functions ..... 7-2
BCD I / O Representation ..... 7-5
Shift_Register ..... 7-8
String Manipulation ..... 7-9
Time Counter ..... 7-11
VME Communication Functions ..... 7-13
Specialized Perform Functions ..... 7-14

**Chapter 8 PID Loops ..... 8-1**

Initializing and Starting PID Loops ..... 8-2
PID Initialization Form ..... 8-2
PID Loop Paramters ..... 8-3
Starting PID Loop Execution ..... 8-5
On-Line PID Loop Tuning ..... 8-6
Program Control of PID Loops ..... 8-6
Program Changes to the Tuning Parameters ..... 8-7
Using the Command and Status Bits ..... 8-7
PID Algorithm and Philosophy ..... 8-8
Simple PID Control ..... 8-9
Complex PID Control ..... 8-9

**Chapter 9 On-Line Features ..... 9-1**

Debug Mode Display Screen ..... 9-2
Controlling the Project ..... 9-2
Logging Data ..... 9-2
Process Simulation ..... 9-2
Download a Project ..... 9-3
Reset and Clear State Engine ..... 9-3
Observing State Engine Values ..... 9-3
Monitor Tables ..... 9-3
View ..... 9-4
Trace ..... 9-4
Display Data ..... 9-4
Faults ..... 9-4
Changing State Engine Values ..... 9-5
Change Variable Data ..... 9-5
Force Inputs and Outputs ..... 9-5
On-Line Hints ..... 9-5

**Chapter 10 Serial Communications Module ..... 10-1**

Overview ..... 10-2
Serial Cables ..... 10-2
SCM Fundamentals ..... 10-3
Installation and Maintenance ..... 10-5
Description ..... 10-5
Installation ..... 10-6
Inserting the SCM ..... 10-6
Configuration ..... 10-7
Battery ..... 10-7
Serial Port Parameters ..... 10-8
Parameter Details ..... 10-8
Changing the Parameter Settings ..... 10-9
CCM2 Protocol Serial Port ..... 10-10
Enabling CCM2 Communication ..... 10-10
CCM Data Types ..... 10-10
Troubleshooting ..... 10-14
Status LED is not ON Steady ..... 10-14
Resets Blinks Port 1 or Port LED ..... 10-14
Serial Communication Problems ..... 10-15

**Appendix A Key Functions ..... A-1**

Function Keys ..... A-1
Hot Keys ..... A-2
Miscellaneous Keys ..... A-2

**Appendix B Language Structure Summary ..... B-1**

Notational Conventions ..... B-1
Language Structure Notational Conventions ..... B-1
Functional Structures ..... B-2
Conditional Structures ..... B-4
Value Expressions ..... B-5

**Appendix C References to the Genius PowerTRAC Block ..... C-1**

PowerTRAC Data Keywords ..... C-2
PowerTRAC Status Bit Keywords ..... C-3

**Appendix D References to Genius High Speed Counter Block ..... D-1**

Counter Register Keywords ..... D-2
Counter Status Bit Keywords ..... D-2
Counter Command Bit Keywords ..... D-3

**Appendix E Errors ..... E-1**

Translation Errors ..... E-1
Runtime Errors ..... E-13
Non-Critical ..... E-13
Critical Errors ..... E-15

**Appendix F Standard Predefined KeyWords ..... F-1**

Conditional Terms ..... F-1
Functional Terms ..... F-2
Operators ..... F-4
Miscellaneous Keywords ..... F-6

**Appendix G Integrating Ladder Logic and/or 'C' Programming ..... G-1**

**Appendix H ECLiPS Specifications ..... H-1**

Content of this Manual ..... iii
We Welcome Your Comments and Suggestions ..... vi

Figure 2-1. 90-70 State Logic Control System ..... 2-2
Figure 2-2. OnTOP Security Form ..... 2-7
Figure 3-1. State Diagrams ..... 3-2
Figure 3-2. Task Description in ECLiPS ..... 3-3
Figure 3-3. Sample Task with Some Elements Labeled ..... 3-5
Figure 3-4. Five State Task Example with a Single State Highlighted ..... 3-6
Figure 3-5. Example ECLiPS State with One complete Statement Highlighted ..... 3-7
Figure 3-6. Highlighted Communication Functions ..... 3-8
Figure 3-7. Example Diagnostic Task ..... 3-13
Figure 4-1. Example ECLiPS State with Four Statements ..... 4-6
Figure 4-2. Statement examples with Functional Terms highlighted ..... 4-6
Figure 4-3. Statement example with Conditional Terms highlighted ..... 4-6
Figure 4-4. Program Scan ..... 4-9
Figure 4-5. Statement Scan ..... 4-10
Figure 4-6. Program Scan with GO Terms ..... 4-11
Figure 4-7. Statement Scan with GO ..... 4-11
Figure 4-8. Using multiple to Tasks to keep an Output ON ..... 4-13
Figure 6-1. ECLiPS Program Hierarchy ..... 6-2
Figure 6-2. Functional Expressions in Bold Type ..... 6-3
Figure 6-3. Conditional Expressions in Bold Type ..... 6-4
Figure 6-4. Examples of Multiple Conditional Term Expressions ..... 6-4
Figure 6-5. Examples of Multiple Functional Term Expressions ..... 6-4
Figure 8-1. PID Initialization Form ..... 8-2
Figure 8-2. PID Tuning Screen ..... 8-6
Figure 8-3. PID Algorithms ..... 8-9
Figure 8-4. Cascaded PID Loops ..... 8-10
Figure 10-1. Serial Port Assignments for Series 90-70 SCM ..... 10-2
Figure 10-2. IBM PC-AT to SCM Cable ..... 10-3
Figure 10-3. Workmaster II or PS / 2 to SCM Cable ..... 10-3
Figure 10-4. SCMs in Series 90-70 Chasis ..... 10-4
Figure 10-5. Serial Communications Module ..... 10-5
Figure 10-6. Sample Logicmaster Configuration Screen ..... 10-7
Figure 10-7. Sample Port Configuration Screen ..... 10-9

Table 6-1. Functional Term Quick Reference List . . . . . 6-5
Table 6-2. Conditional Term Quick Reference List . . . . . 6-10
Table 8-1. PID Loop Parameters . . . . . 8-4
Table 8-2. PID Command and Status Bits . . . . . 8-7
Table 10-1. Slot Number to Serial Port Number Correlation . . . . . 10-3
Table 10-2. Slot Number to Serial Port Number Correlation . . . . . 10-6
Table 10-3. Serial Port Parameters . . . . . 10-8
Table 10-4. State Engine Data Types and CCM References . . . . . 10-11
Table 10-5. Digital Point Names . . . . . 10-11
Table 10-6. Internal Flag Names . . . . . 10-11
Table 10-7. Analog Channel Names . . . . . 10-11
Table 10-8. String Variable Names . . . . . 10-12
Table 10-9. Integer Variable Names . . . . . 10-12
Table 10-10. Floating Point Variable Names . . . . . 10-12
Table 10-11. PID Loop Names . . . . . 10-12
Table 10-12. Task and State Names . . . . . 10-12
Table 10-13. PID Parameter Table . . . . . 10-13
Table 10-14. SCM Specifications . . . . . 10-15
Table 10-15. Standards . . . . . 10-15
Table G-1. State Engine Register Usage . . . . . G-2

# GE Intelligent Platforms North America, Inc.
# Software License Agreement

GFK-0731B

YOU SHOULD CAREFULLY READ THE FOLLOWING TERMS AND CONDITIONS BEFORE OPENING THIS PACKAGE. OPENING THIS PACKAGE SIGNIFIES YOUR ACCEPTANCE OF THESE TERMS AND CONDITIONS. IF YOU DO NOT AGREE WITH THEM, YOU SHOULD PROMPTLY RETURN THE PACKAGE UNOPENED ALONG WITH ANY OTHER ITEM THAT WAS INCLUDED IN THE SAME CATALOG NUMBER FOR FULL CREDIT.

You, as the Customer, agree as follows:

## 1. DEFINITIONS

"Application Software" shall mean those portions of the Licensed Software, in object code form only, created by GE.

"Designated Computer" shall mean the one (1) computer upon which Customer shall run the Licensed Software.

"Licensed Software" shall mean the Application Software plus any other software, in object code form only, supplied by GE pursuant to this Agreement. The Licensed Software may include third party software, including but not limited to operating systems, licensed to GE. If no operating system software is included in the software provided under this Agreement, you must make provision for any required operating system software licenses.

## 2. LICENSE

2.1 Except as provided in section 2.2 below, you are granted only a personal, non–transferable, nonexclusive license to use the Licensed Software only on the Designated Computer. You may copy the Licensed Software into machine readable form for backup purposes in support of your use of the Licensed Software on the Designated Computer, limited to one copy. No other copies shall be made unless authorized in writing by GE. You may not reverse compile or disassemble the software. The Licensed Software, comprising proprietary trade secret information of GE and/or its licensors, shall be held in confidence by Customer and protected from disclosure to third parties. No title to the intellectual property is transferred. You must reproduce and include all applicable copyright notices on any copy.

2.2 If you are an authorized GE distributor or an Original Equipment Manufacturer who incorporates the Licensed Software into your equipment for sale to an end user, you may transfer the Licensed Software to an end user provided that the end user agrees to be bound by the provisions of this Agreement.

2.3 GE licensors having a proprietary interest in the Licensed Software shall have the right to enforce such interests, including the right to terminate this Agreement in the event of a breach of its terms pertaining to such proprietary interests.

2.4 EXCEPT AS PROVIDED IN SECTION 2.2 ABOVE, IF YOU TRANSFER POSSESSION OF ANY COPY OF THE LICENSED SOFTWARE TO ANOTHER PARTY WITHOUT WRITTEN CONSENT OF GE YOUR LICENSE IS AUTOMATICALLY TERMINATED. Any attempt otherwise to sublicense, assign or transfer any of the right, duties or obligations hereunder is void.

2.5 If the Licensed Software is being acquired on behalf of the U.S. Government, Department of Defense, the Licensed Software is subject to "Restricted Rights", including the legend to be affixed to the software as set forth in DOD Supplement to the Federal Acquisition Regulations (DFAR's) paragraph 252.227–7013(c)(1). If software is being acquired on behalf of any other U.S. Government entity, unit or agency, the Government's rights shall be as defined in paragraph 52.227–19(c)(2) of the Federal Acquisition Regulations (FAR's).

## 3. WARRANTY

3.1 GE warrants that the Application Software will be in substantial conformance with the specifications in the manual pertaining thereto as of the date of shipment by GE. If, within ninety (90) days of date of shipment, it is shown that the Application Software does not meet this warranty, GE will, at its option, either correct the defect or error in the Application Software, free of charge, or make available to Customer satisfactory substitute software, or, as a last resort, return to Customer all payments made as license fees and terminate the license with respect to the Application Software affected. GE does not warrant that operation of the Application Software will be uninterrupted or error free or that it will meet Customer's needs. All other portions of the Licensed Software are provided "as is" without warranty of any kind.

3.2 WITH RESPECT TO THE SOFTWARE WHICH IS THE SUBJECT OF THIS AGREEMENT, THE FOREGOING WARRANTIES ARE EXCLUSIVE AND ARE IN LIEU OF ALL OTHER WARRANTIES WHETHER WRITTEN, ORAL, IMPLIED OR STATUTORY NO IMPLIED OR STATUTORY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE SHALL APPLY.

## 4. LIMITATION OF LIABILITY

4.1 IN NO EVENT, WHETHER AS A RESULT OF BREACH OF CONTRACT, BREACH OF WARRANTY, TORT (INCLUDING NEGLIGENCE) OR OTHERWISE SHALL GE OR ITS SUPPLIERS BE LIABLE FOR ANY SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PENAL DAMAGES INCLUDING, BUT NOT LIMITED TO, LOSS OF PROFIT OR REVENUES, LOSS OF USE OF THE LICENSED SOFTWARE OR ANY PART THEREOF, OR ANY ASSOCIATED EQUIPMENT, DAMAGE TO ASSOCIATED EQUIPMENT, COST OF CAPITAL, COST OF SUBSTITUTE PRODUCTS, FACILITIES, SERVICES OR REPLACEMENT POWER, DOWN TIME COSTS, OR CLAIMS OF CUSTOMER'S CUSTOMERS AND TRANSFEREES FOR SUCH DAMAGES EVEN IF GE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

4.2 EXCEPT AS PROVIDED IN SECTION 5, INDEMNITY, IN NO EVENT, WHETHER AS A RESULT OF BREACH OF CONTRACT OR WARRANTY, TORT (INCLUDING NEGLIGENCE) OR OTHERWISE, SHALL GE LIABILITY TO CUSTOMER FOR ANY LOSS OR DAMAGE ARISING OUT OF, OR RESULTING FROM THIS AGREEMENT, OR FROM ITS PERFORMANCE OR BREACH, OR FROM THE LICENSED SOFTWARE OR ANY PART THEREFORE, OR FROM ANY SERVICE FURNISHED HEREUNDER, EXCEED THE QUOTED CHARGES FOR THE LICENSED SOFTWARE. ANY SUCH LIABILITY SHALL TERMINATE UPON THE TERMINATION OF THE WARRANTY PERIOD AS SET FORTH IN SECTION 4.

4.3 If GE furnishes Customer with advice or other assistance which concerns Licensed Software or any portion thereof supplied hereunder or any system or equipment on which any such software may be installed and which is not required pursuant to this Agreement, the furnishing of such advice or assistance will not subject GE to any liability, whether in contract, warranty, tort, (including negligence) or otherwise.

4.4 The products to be licensed or sold hereunder are not intended for use in any nuclear, chemical or weapons production facility or activity, or other activity where failure of the products could lead directly to death, personal injury or severe physical or environmental damage. If so used, GE disclaims all liability for any damages arising as a result of the hazardous nature of the business in question, including but not limited to nuclear, chemical or environmental damage, injury or contamination, and Customer shall indemnify, hold harmless and defend GE, its officers, directors, employees and agents against all such liability, whether based on contract, warranty, tort (including negligence), or any other legal theory, regardless of whether GE had knowledge of the possibility of such damages.

## 5. INDEMNITY

5.1 GE warrants that the Application Software shall be delivered free of any rightful claim for infringement of any United States patent or copyright. If notified promptly in writing and given authority, information and assistance, GE shall defend, or may settle, at its expense, any suit or proceeding against Customer so far as based on a claimed infringement which would result in a breach of this warranty and GE shall pay all damages and costs awarded therein against Customer due to such breach. In case the Application Software is in such suit held to constitute such an infringement and its use is enjoined, GE shall, at its expense and option, either procure for Customer the right to continued use, or replace same with a non–infringing product or part, or modify the Application Software so that it becomes non–infringing, or remove the software and refund the license charge pertaining thereto (less reasonable depreciation for any period of use) and any transportation costs separately paid by Customer. The foregoing states the entire liability of GE for patent and copyright infringement by the Licensed Software or any part thereof.

5.2 The indemnity under the preceding paragraph shall not apply to any use of Application Software in conjunction with any other product in a combination not furnished by GE as a part of this transaction. As to any such use in such combination, GE assumes no liability whatsoever for patent and copyright infringement and Customer will hold GE harmless against any infringement claims arising therefrom.

## 6. TERM AND TERMINATION

6.1 You may terminate the license granted hereunder at any time by destroying the Licensed Software together with all copies thereof and notifying GE in writing that all use of the Licensed Software has ceased and that same has been destroyed.

6.2 GE, upon thirty (30) days notice, may terminate this Agreement or any license hereunder if Customer fails to perform any obligation or undertaking to be performed by it under this Agreement or if Customer attempts to assign this Agreement without the prior written consent of GE. Within twenty (20) days after any such termination of this Agreement, Customer shall certify in writing to GE that all use of the Licensed Software has ceased, and that same has been returned or destroyed, in accordance with GE instructions.

6.3 Sections 4, 6 and 7 of this Agreement shall survive any expiration or termination and remain in effect. Termination of this Agreement or any license hereunder shall not relieve Customer of its obligation to pay any and all outstanding charges hereunder nor entitle Customer to any refund of such charges previously paid.

## 7. EXPORT

7.1 If you intend to export (or reexport), directly or indirectly, the software products or technical information relating thereto supplied hereunder or any portion thereof, it is your responsibility to assure compliance with U.S. export control regulations and, if appropriate, to secure any required export licenses in your own name.

## 8. GENERAL

8.1 This Agreement shall be governed by the laws of the State of Virginia, without regard to its conflict of law provisions. The provisions of the United Nations Convention on the International Sale of Goods shall not apply to this Agreement.

Should you have any questions concerning this Agreement, you may contact GE by writiing to:
**GE, P.O. Box 8106, Charlottesville, VA 22906.**

YOU ACKNOWLEDGE THAT YOU HAVE READ THIS AGREEMENT, UNDERSTAND IT AND AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS. YOU FURTHER AGREE THAT IT IS THE COMPLETE AND EXCLUSIVE STATEMENT OF THE AGREEMENT BETWEEN US AND SUPERSEDES ANY PROPOSAL OR PRIOR AGREEMENT, ORAL OR WRITTEN, AND ANY OTHER COMMUNICATIONS BETWEEN US RELATING TO THE SUBJECT MATTER OF THIS AGREEMENT. FURTHER, NO CHANGE OR AMENDMENT TO THIS AGREEMENT SHALL BE EFFECTIVE UNLESS AGREED TO BY WRITTEN INSTRUMENT SIGNED BY A DULY AUTHORIZED REPRESENTATIVE OF GE.

# Chapter 1 Preliminaries

This chapter describes some details that should be completed before using this product. Please take the time to review all of the items in this short chapter.

## Items Included with ECLiPS

Check your ECLiPS package that it has the following items:

1. ECLiPS Disks and this Manual
2. ECLiPS Hardware Key
3. OnTOP Disk and Manual
4. Logicmaster 90-70 Demo Version Disk
5. Mini-Converter Kit, Serial Cable and Adapters

If any of these parts is missing, contact your local distributor.

## Getting Help

There are three ways to get help:

1. ECLiPS Help System. ECLiPS has a built in help system that can always be accessed by pressing the key on your keyboard marked <F1>. This help system is context sensitive meaning that ECLiPS provides the helpful information you need based on the location of the cursor on the screen or the highlighted menu option at the moment you ask for help.

   More information about using the ECLiPS help system can be found in the reference chapter of this manual.

2. ECLiPS Reference Manual. The reference chapter of this manual contains helpful information organized by the command, function or procedure name. Use the main index at the back of this manual or the reference chapter index at the beginning of the reference chapter to locate information.

3. GE has personnel specially trained throughout the country to provide customer support for ECLiPS and other State Logic products that work together with standard GE control products. Call GE Intelligent Platforms technical support line at 1-800-828-5747.

# How to Use this Manual

This section explains how to use this manual. First is a description of each of the chapters followed by an explanation of how the chapters are organized. Finally is an explanation of how to take advantage of this organization to find the information needed.

## Brief Description of the Manual Chapters

### 1. Preliminaries

Preliminaries is the chapter you are now reading.

### 2. Setup

This chapter first shows the general architectural view of the 90-70 State Logic Control System then explains all of the setup procedures required to run the ECLiPS programming software and operate the 90-70 State Engine.

### 3. State Logic Control Theory

This section provides some basics about the underlying concepts and philosophy of State Logic Control. Regardless of what you may already know about State Logic, **it is extremely important that you read this section carefully.**

### 4. Creating a State Logic Control Program

This chapter explains how a control application is programmed using ECLiPS.

### 5. Program Features

This chapter shows the ECLiPS features devoted to creating and modifying State Logic Programs.

### 6. Program Structure

The Program Structure chapter explains how to use the functional words in the State Logic Program Language.

### 7. Perform Functions

The Perform Functions are a series of advanced State Logic Language features. This chapter explains how to implement them in a program

### 8. PID Loops

This chapter explains how to use PID Loops in the State Logic Program.

## 9. On - Line Features

The On - Line Features are designed to view and modify the information being handled by the State Engine. This chapter explains what features are available and how to use them.

## 10. Serial Communications Module

The Serial Communication Module or SCM is an add on module that is used for serial communication to serial devices or a host computer. This chapter will explain how to use the SCM and what features are available with it.

## Appendix A Function Keys

Appendix A contains a summary of all of the function keys.

## Appendix B Language Structural Summary

Contains a list of all language structures

## Appendix C PowerTRAC Data Blocks

Describes the use of PowerTRAC Genius I/O Blocks.

## Appendix D High Speed Counter Data Blocks

Describes the use of the Genius block high Speed Counter.

## Appendix E Error Messages

Contains a list of all of the ECLiPS generated error messages.

## Appendix F Predefined Keywords

List of Keywords that come defined in ECLiPS.

## Appendix G Ladder Logic and/or 'C' Programming

How to integrate different programming methods with State Logic.

## Appendix H ECLiPS Specificications

The ECLiPS programming limits.

## Manual Organization

To find specific information in this manual use the manual organization. First determine whether the topic falls into one of these categories:

**Setup**

**Programming**

**On - Line**

**Serial Communications**

Then go to the chapter or group of chapters that deals with that topic.

The **setup** options are covered in the chapter, **Setting Up the 90-70 State Logic Control System**. This chapter explains the overall system architecture in addition to how to set up and configure the components of the system. Overall system considerations are covered in this chapter.

The next group of chapters have information on the details of creating a State Logic control **program**. The chapters proceed from fundamental, **State Logic Theory,** to most complex, **PID Loops.** The **State Logic Theory** chapter is a very important chapter for first time users is. *State Logic is very different from any other type of programming language, so even if you have extensive experience with procedural languages such as C, BASIC or FORTRAN or other languages designed for control such as Relay Ladder Logic or Boolean, read this chapter.*

The programming chapters are followed by the chapter on the ECLiPS **On-line Features** chapter.

The last chapter describes the Serial Communications Module (SCM) used to provide serial interface to the system.

Use this grouping of the chapters to locate the information needed. Also use the **Table of Contents** and the **Index** to help locate specific topics.

# Chapter 2 Setting Up the 90-70 State Logic Control System

This chapter explains how to set up the 90-70 State Logic system. The first section describes the State Logic Control System architecture. The following sections explain how to setup the different components of the State Logic Control System.

# System Overview (Architecture)

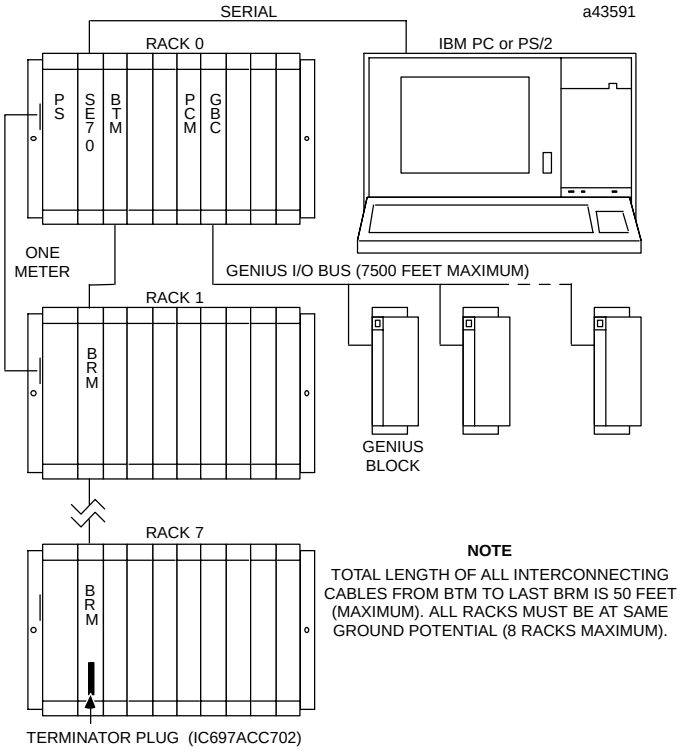


**Figure 2-1. 90-70 State Logic Control System**

The Series 90-70 State Logic Control system has two main components:

**Series 90-70 State Engine and PLC System**

**The PC Programmer and Software to Interface to the State Engine**

The State Logic PLC System includes the State Engine CPU module which executes the control program, the PLC rack and power supply, and I/O modules. The State Engine module comes with a memory protection key switch, memory battery backup, four indicator LED's, run switch, and an RS-422/485 serial port. For more information on the State Engine module look for CPU information the GE manuals GFK-0262 and GFK-0265.

The memory protect key switch prevents the program from being changed when switched ON. An attempt to download a program or make on-line changes will fail when the switch is ON.

The PC Programmer and interface software is an IBM PC compatible or PS/2 which runs one of the software packages ECLiPS, OnTOP, or Logicmaster 90 Demo version.

# ConnectingECLiPS, OnTOP, and Logicmaster 90 to the State Engine

ECLiPS, OnTOP, and Logicmaster 90 are all software packages that run on an IBM PC compatible or PS/2. ECLiPS is used to create the State Logic program and perform on-line operations, OnTOP does only on-line operations and Logicmaster 90 loads the State Engine operating system and configures the PLC system. Each of these packages connect to the SNP serial port located on the State Engine.

Use the mini-converter kit supplied with ECLiPS to connect the State Engine to the computer running any of the software packages. Plug the 15-pin to 9-pin adapter, part number HE693SNP232, into the serial port on the State Engine. Connect the serial cable to the computer running ECLiPS, OnTOP, or LogicMaster to either COM port 1 or 2. Use part number HE693XTADP to convert a 25 pin COM port to 9 pin for the serial cable.

ECLiPS, Logicmaster, and OnTOP use the same serial ports and the same cables to connect to the State Engine. When changing from one package to another, no changes are required in the serial cable or port. ECLiPS and OnTOP can use COM 1, or 2. LogicMaster can be run using COM 1, 2, 3, or 4.

To use ports other than COM1 for Logicmaster 90 set the environment variable PLC_COM_PORT by entering a line such as

**set PLC_COM_PORT=2**

in the AUTOEXEC.BAT file.

For more information on using Logicmaster 90 refer to the Logicmaster manuals.

# Setting Up ECLiPS

## Hardware Requirements

1. IBM PC compatible or PS/2
2. 640K RAM - Expanded and/or Extended Memory Optional
3. DOS version 3.1 or higher
4. 3 MB of Hard Disk Space
5. 3.5 inch floppy disk drive
6. Any printer (Optional)
7. Color or monochrome monitor
8. Serial Port

## ECLiPS Installation

To install ECLiPS, insert disk 1 into drive A or B and make this drive the current logged drive. Type **INSTALL** and hit <Enter>. Be ready to specify the path (drive and directory) where ECLiPS is to be installed. Follow the instructions for inserting other disks. The installation program displays a message when the installation is complete.

To run ECLiPS, make sure that the directory where ECLiPS is installed is the current directory. Now type **ECLIPS** to start the program.

There should be lines in the CONFIG.SYS file to set the FILES and BUFFERS to at least the following minimums:

```
FILES=20
BUFFERS=20
```

## Hardware Key

ECLiPS is not copy protected and may be installed repeatedly, but the license agreement specifies that ECLiPS is to be used on only one computer at a time.

ECLiPS requires a hardware key for normal operations. ECLiPS does function in a limited manner when the hardware key is not present. Operation of ECLiPS without the hardware key is allowed by the license only for demonstration purposes to evaluate the operation of the product. DO NOT attempt to use ECLiPS for control operations without having the key installed.

To install the hardware key, plug it into the parallel port (LPT1) and tighten the security screws. The port must be a fully compatible IBM Printer Adapter Card. The key does not restrict or change the use of the parallel port. Any device that normally connects to the parallel port may be connected to the key and all information is transmitted just as if the key were not present.

If another software product requiring a similar hardware key is installed on the same computer as ECLiPS, plug the ECLiPS key directly into the parallel port and connect the other key to the ECLiPS key.

If there is a need to move ECLiPS to another computer the distribution disks may be used to install ECLiPS on the other computer and the hardware key moved to the new computer. To return to the original computer, move the key to that computer.

## Configuring ECLiPS Memory Usage

ECLiPS makes use of Expanded and Extended memory if they are available with appropriate memory managers installed. By using expanded and/or extended memory, ECLiPS can be configured for more States and Variables and also executes faster. The Expanded memory and memory manager must be version LIM 3.2 or higher, and the Extended memory and memory manager must be version XMS 2.0 or higher.

EMM386.SYS can be used for Expanded memory and HIMEM.SYS can be used for Extended memory. Two megabytes of memory is enough to provide the maximum capacity and speed for ECLiPS operations.

The first ECLiPS screen, the Main Menu, shows the **SETUP** option which is used to configure the ECLiPS system. The two parameters that can be selected are; maximum number of States and the maximum number of I/O and variable names. The number of names can be from 500 to 3000 selected in increments of 500. The number of States is 500 to 3000 in increments of 500.

The maximum limits cannot be selected for both the number of States and number of Names for I/O and variables. The maximum total number for States plus names is 4500. If the ECLiPS computer does not have enough memory for the selected limits, ECLiPS displays an out of memory error message and allows the Setup limits to be reselected.

There are two reasons to select the smallest size limits for the ECLiPS setup options. First ECLiPS requires less free conventional memory with lower defined limits. The second reason is that ECLiPS does some operations faster when using the smaller limits.

If there is not enough conventional memory available to load ECLiPS, the Setup screen is displayed to re configure ECLiPS for a smaller number of States and/or variables. After these quantities are changed, ECLiPS is restarted.

Another way to deal with the problem of not enough memory to run ECLiPS, is to free up conventional memory in your PC by eliminating device drivers and Terminate and Stay Resident (TSR) programs by changing your CONFIG.SYS and AUTOEXEC.BAT files. If you are using DOS 5 or better, loading DOS HIGH also frees up additional conventional memory. To use the maximum settings for either the number of States or Variables, DOS must be loaded into the high memory area.

## Loading Logicmaster Hardware Configuration

ECLiPS can retrieve the Logicmaster hardware configuration. This function allows the configuration to be viewed from ECLiPS using rack displays similar to Logicmaster. This display allows the user to view and modify; a) CPU configuration, b) I/O names and assignments, and c) Comm Port Configurations.

To make the Logicmaster 90 configuration information available to ECLiPS, use the "Retrieve Logicmaster Configuration Data" option from the DEFINE menu. ECLiPS asks for the path name of the Logicmaster folder that stores the data. Any valid hardware configuration can be used to create the configuration data.

After retrieving the configuration information from Logicmaster, it may be accessed by selecting the "System Configuration" option from the DEFINE menu. This option displays a rack of the 90-70 PLC system similar to the display in Logicmaster. The cursor can be moved through the form with the arrow keys. Using <PageUp> and <P ageDown> allows viewing of additional racks. Placing the cursor on top of a module and pressing enter will bring up a menu for that type of module.

## State Engine Configuration Options

To configure the CPU, highlight the CPU slot and press enter. A menu of choices is provided. Choose View / Modify Configuration, a form of options is displayed. The following parameters can be set:

Select Whether to Run Program on Power Up

Select Runtime Fault Response

Change the Name of the ECLiPS Port

The information from this form will be downloaded to the State Engine each time the State program is downloaded. If a change is made to the configuration menu, the project will need to be translated and downloaded again for the changes to take effect.

**I/O modules** can be configured using the DISPLAY "System Configuration" option. Selecting the I / O device of choice will display a menu. Choosing the I / O Reference option gives access to a form that displays all of the I / O points associated with that module. Any I / O names that have been defined will be shown at the assigned I / O point. Writing a name to an I / O point will cause that name to be defined.

When a **Serial Communications Module (SCM)** is selected, the form that is displayed will show all of the configuration parameters for the serial ports that module provides. See the chapter on the Serial Communications Module for more information.

When a **Genius Bus Controller** is selected, each block on the bus can be displayed. The circuits for I / O blocks can be named by selecting the block. Any %G global data discrete bits can also be named from this display.

## Security Setup

Both ECLiPS and OnTOP can be configured to require passwords to execute. ECLiPS uses only one password, but OnTOP has four levels of password protection.

## ECLiPS Security

After being configured to require a password, ECLiPS cannot be started without entering that password. Use this security protection to restrict access to the State Engine and the ECLiPS program files. The password protection only limits entering the ECLiPS package and does not limit access to the State Engine by another installation of ECLiPS.

Another aspect of the State Logic system that provides security for the State Engine program is that ECLiPS cannot connect with the State Engine unless it can access the same version of program files as those that are running in the State Engine. No one can run or inspect the State Engine program without having a copy of the correct version of the project files.

## OnTOP Security System

The OnTOP Security System is designed for machine operation safety. Through this feature unauthorized access to the control of the machine is prevented. The security system allows up to 4 levels of security for OnTOP. Levels 1 - 3 are customized to allow or disallow access to six functions. Level number 4 allows unlimited access to all of the features in OnTOP, but the State Logic Program cannot be changed.

To initialize the Security System, run ECLiPS from the same drive as OnTOP. Select "Security" from the ECLiPS Program Mode menu. Often, ECLiPS is uninstalled from the drive after the program security system is initialized and the program is downloaded to the controller.

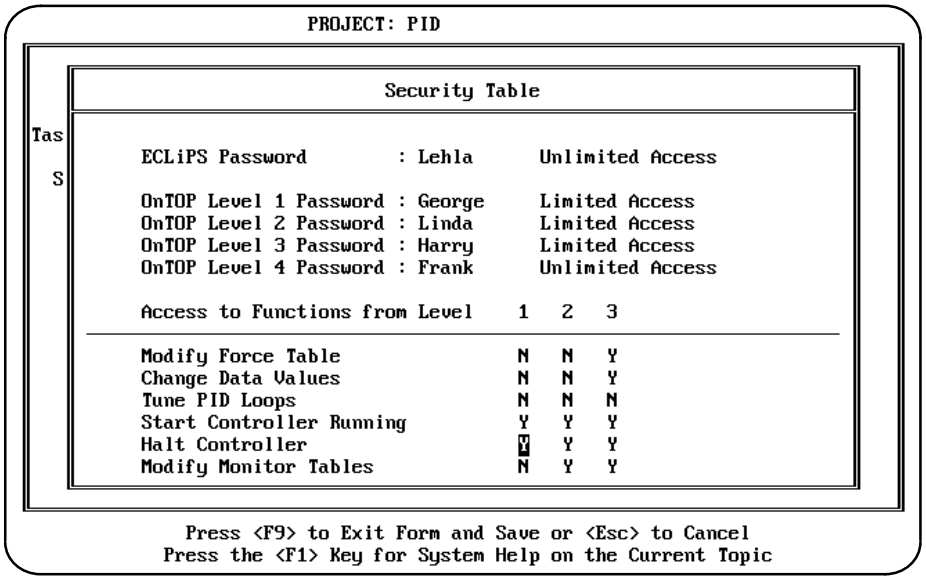
PROJECT: PID
Security Table
Tas
S
ECLiPS Password : Lehla Unlimited Access
OnTOP Level 1 Password : George Limited Access
OnTOP Level 2 Password : Linda Limited Access
OnTOP Level 3 Password : Harry Limited Access
OnTOP Level 4 Password : Frank Unlimited Access
Access to Functions from Level 1 2 3
Modify Force Table N N Y
Change Data Values N N Y
Tune PID Loops N N N
Start Controller Running Y Y Y
Halt Controller Y Y Y
Modify Monitor Tables N Y Y
Press <F9> to Exit Form and Save or <Esc> to Cancel
Press the <F1> Key for System Help on the Current Topic

**Figure 2-2. OnTOP Security Form**

Use the arrow keys to move the highlighted cursor to the OnTOP Level 1 Password block in the security form. Enter the password. Move the cursor to Level 2, 3, 4 and enter the appropriate passwords. Move the cursor to the Function Access Table and enter a "Y" or an "N" to activate or deactivate the function listed for each level of security access. Also use this form to make any changes to an existing security system.

# Setting Up OnTOP

## Installation

To install OnTOP, place the disk into a floppy drive making that drive the currently logged drive type INSTALL. There is no copy protection for OnTOP but the license agreement specifies that each license allows it to execute on only one computer at a time.

### Using ECLiPS Program Files

OnTOP uses the program files created by ECLiPS for on-line interaction with the State Engine. These files provide OnTOP with the actual State Logic program and variable definitions so that information about the operation of the State Engine can be displayed.

The files that have variable definitions use the project name with extension PRJ. If the project name were PRESS then the definitions would reside in the file PRESS.PRJ.

The files that contain the program information have extensions TG0 through TGF. The ECLiPS program may have up to sixteen Task Groups, the first task group is saved in file with extension TG0 and the second in the file with extension TG1 etc. up to TGF.

These ECLiPS files must be on a disk that OnTOP can access before OnTOP can even connect with the State Engine. If OnTOP cannot find the files to match the State Engine program it asks for another path that it can search to find the files.

## Setting Up Logicmaster 90

To install the Demo version of Logicmaster 90, place disk 1 in a floppy drive and type INSTALL at the DOS prompt, then follow the directions displayed on the screen. Type LM90DEMO to start this version of Logicmaster running. Other setup operations and directions for the use of Logicmaster can be found in the Logicmaster manuals.

## Setting Up the State Engine

There are two main steps to setting up the State Engine, loading the operating system and configuring the system. Both operations are accomplished with Logicmaster 90-70 Demo version.

### Loading the State Engine Operating System

The 90-70 State Engine hardware is the 90-70 CPU module. The 90-70 State Engine is the CPU of the 90-70 PLC System and therefore must be inserted into slot 1 of rack 0 of the system.

The operating system for the State Engine is located on the State Engine Operating System disk which accompanies the State Engine. To load the operating system, connect Logicmaster to the State Engine with the serial cable and adapter, then perform the following steps:

1. Type LM90DEMO to start Logicmaster demo version running.
2. Select the Programmer Package, <F1>.
3. Place the State Engine operating system disk in a floppy drive.
4. When Logicmaster asks for a folder to use, enter A:\ENGINE. This folder may also be selected by pressing <F8> then <F1> to name a folder to be used.
5. Press <F9> for the Utility option.
6. Press <Alt + M> repeatedly until Logicmaster displays ON-LINE in the mode display in the bottom center of the screen.

7. Press <F2> to load the operating system into the State Engine
8. Press <Enter> and then Y when told that items will be overwritten and asked whether to continue.
9. Press <ALT + R> to start the State Engine Operating System executing.
10. When the store is complete, press <ESC> repeatedly and answer Yes when asked "do you want to exit Logicmaster" to exit LM90DEMO.

    To combine ladder logic and / or 'C' programming with the State Logic program, copy the State Engine folder to another folder and add the other programming instructions to the State Engine program block . To add other types of programming to the State Engine it may be necessary to use the full Logicmaster version. The demo version of Logicmaster provides a limited number of rungs and program blocks which may limit how much other programming can be added .

## Configuring the State Engine Hardware System

The 90-70 State Engine is completely software configurable, no jumpers or switches are used. The complete system can be configured using the demo version of Logicmaster 90-70 included with ECLiPS. The items that require configuration are; a) the memory setup for the State Engine (system CPU), b) the types of modules inserted into each slot, and c) the number and types of I / O racks used. To configure the system follow these steps:

1. Type LM90DEMO to start Logicmaster.
2. At the Logicmaster main menu, select the Configuration Package, <F2>.
3. At the Configuration package main menu, select I / O Configuration <F1>.
4. Select the appropriate CPU for the system. Highlight CPU slot and press <F10>, a blank space for the CPU model number is displayed, press <F1> for a list of possible CPU types, select the appropriate CPU number by placing the cursor on it and pressing enter

| State Engine Part Number | Configuration Part Number |
|---|---|
| IC697CSE784 | IC697CPU782 |
| IC697CSE924 | IC697CPU924 |

    Memory daughter boards for the 782 processor need to be added by pressing <F9> followed by <F1>, and selecting the correct board from the list. Press <ESC> to save CPU configuration and return to rack drawing.

5. Configure each of the other slots used in the system. Refer to Logicmaster manuals for configuration information. After completing the configuration press <ESC> to save the configuration and return to the Configuration Main Menu.
6. From the Configuration Main Menu, Press <F2> for CPU Configuration then choose <F4> for Memory Allocation options.
7. Make sure Analog Input, %AI, and Analog Output, %AQ, are both set to 1024 and Register memory, %R, is set to 16384. Press <ESC> twice to save memory settings and return to Configuration Main Menu.
8. From the Configuration Main Menu, Press <F9> for the UTILITY functions.

9. Press <ALT + M> to change the mode to ON-LINE, you may have to press <ALT + M> twice. If the State Engine is in Run Outputs/Enabled mode the scan time will be displayed at the bottom of the screen. If the State Engine is in run mode press <ALT + R> to take it out of run mode.
10. Press <F2> to store the configuration to the State Engine. Press <ALT + R> to put the State Engine back into run mode. Press <ESC> twice to exit the utility menu and return to the Configuration Main Menu.
11. To set the State Engine system clock, press <F2> from the Configuration Main Menu. Press <F1>, to set the State Engine (Called PLC clock) time. Press <Alt + M> to be ON-LINE. LogicMaster will load the correct time in the clock. Press <ESC> twice to return to Configuration Main Menu.
12. Press <ALT + R> to start the State Engine Operating System executing.
13. From the Configuration Main Menu, Press <ESC> and answer Y to exit to LM90DEMO Main Menu. Press <ESC> to exit LM90DEMO.

# Communicating with the State Engine

There are three ways to communicate with the State Engine:

1. Using Programmer Software (ECLiPS, OnTOP, or Logicmaster)
2. Using the Serial Ports (Serial Communications Module)
3. Using the 90-70 MMS Ethernet Module

## Communicating with the Programmer Software Packages

All of the programmer packages communicate with the 90-70 State Engine through the State Engine serial port. A previous section in this chapter describes the details of connecting these packages to the State Engine.

ECLiPS communicates to the State Engine to download programs and display on-line information. OnTOP can also download programs and display on-line information. Logicmaster 90 loads the State Engine Operating System and configures the State Engine memory usage and I/O modules.

## Serial Communication

Serial communication between the State Engine and other devices is accomplished through the Serial Communication Module or SCM. One of the SCM ports can also be used for CCM protocol communications to connect SCADA and other host computer programs to the State Engine. For more information on the SCM see the chapter devoted to the Serial Communications Module.

## EtherNet Communication

Communication between the State Engine and a host computer with the EtherNet protocol can be accomplished with the EtherNet Module. Refer to the GE manuals for information about the Ethernet module. Much of the State Engine information accessible by the Ethernet module resides in register locations. See the Appendix G of this manual for a register map of which registers are used by the State Engine to store different types of State Logic information. There are special functions available for accessing data that is not available in the registers.

# Chapter 3

# State Logic Control Theory

This chapter has two main topics. The first part discusses State Logic Control Theory and how it differs from traditional control models. The second part discusses the ECLiPS implementation of State Logic Control.

# State Logic Control Theory

State Logic Control has its roots in Finite State Machine Theory, developed by nineteenth century mathematicians. Because its philosophy is a natural fit to real-time systems, Finite State Machines have become the strategy of choice in disciplines, such as electronics and compiler design. This theory has recently been gaining wide spread acceptance in industrial control industry with all major controls producers offering State Logic based control products.

## The Concept of Finite States

The basic concept of State Logic is that a process can be defined as a sequence of States. Each State is defined by two components, actions that occur while that State is active and the **transitions** to other States.

In the control world, **actions** are turning ON digital outputs, setting variable and analog output values, sending messages to an operator, etc.

**"Turn ON Mixer_Motor."**

is an example of an ECLiPS program line describing the **action** of a State.

Transitions are a little more complicated since they are themselves defined by two components, the condition controlling the transition and the target State.

**"If Part_In_Place switch is ON, go to Start_Conveyor State."**

is an ECLiPS program line representing a **transition** of a State. In the control world conditions controlling **transitions** are the status of the digital inputs, the values of variables and analog inputs, elapsed time, etc. The target State is the one which becomes active when the condition is true.

A sequence of states can describe any control application. In pure Finite State Machine science these sequences are each called State Machines. ECLiPS calls each such sequence a TASK.

It is traditional to diagram Finite State Machines with circles and arrows. The **actions** of a State are written inside the circles. The arrows show the **transitions** with the condition component of the transition written next to the arrow. The following unlabeled State diagrams show two simple Finite State Machines or Tasks.

a80001

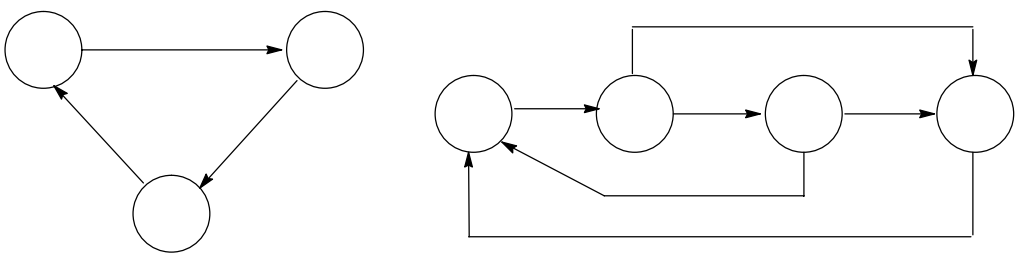

**Figure 3-1. State Diagrams**

The Task may transition from one State to any other State in the Task depending on how the State instructions are specified by the system designer. The target State of all transitions is always pre-defined. A State description may describe several different State transitions based on differing input information. Each Task is always in one and only one State at any time, and the transfer from one State to another does not consume any time.

```
Project: CHEMICAL PROCESS
Task: Make_Compound_5

State: PowerUp
  If the Manual_Switch is on and Start_Pushbutton
  is pressed go to the Add_Water State.
  Go to Add_Water if Auto_Switch is on.

State: Add_Water
  Run Pump_1 until Tank_Guage equals 35 gallons,
  then go to the Add_Chemicals State.

State: Add_Chemicals
  When the Chemical_Management Task is in the Mixing State,
  turn Pump_2 on.  When Tank_Guage equals 39 gallons,
  send "Tank Filled" to operator_panel and go to the Mixing State.

State: Mixing
  If hour is past 8 AM, start the exhaust_system.
  Start Main_Mixer. If 20 seconds pass and the Mixer_Monitor
  s less_than 100 rpms, go to the Wait_3 State.
  Go to the Cooking State after 90 seconds.

State: Wait 3
  Write "PROCESS SHUT DOWN BECAUSE MATERIAL IS TOO THICK".
  Go to PowerUp State when Reset_Button is pushed.
```

**Figure 3-2. Task Description in ECLiPS**

An **IMPORTANT** point is that Finite State Theory does not create or invent TASKS. TASKS are already an inherent part of every process to be controlled. Programming with a state control language is merely the act of describing the process.

## What Makes State Control Logic Different

Both State Logic and traditional methods of control test the condition of the inputs and internal data to decide how to control a system. The fundamental difference of State Logic is its inherent ability to also use the current condition (State) of the process in making control decisions. Traditional methods of control artificially simulate different States with internal contacts or data values. Consider the following States:

```
State: Ready_For_Cutting
  Turn on the Cutter_Ready_Light.
  When the Cut_Push_Button is pressed, go to the Engage_Cutter State.
```

```
State: Engage_Cutter
  Start the Cutter_Blade.
  When the Cut_Complete_Detector is tripped, go to the Raise_Blade State.
```

These States describe a situation where the only time that the cutter should be activated from the push button, is when the machine is ready for the cutting operation. State Logic inherently allows the system designer to take the State of the process into the decision. By making the only transition to the Engage_Cutter State be in the Ready_For_Cutting State, the designer limits the time that the Cut_Push_Button has any affect on the Cutter_Blade.

In these States the only time the cutter is started is when the Engage_Cutter State is active. Traditional approaches allow for ingenious methods to simulate the States of the process to protect from an inadvertent pressing of the Cut_Push_Button at the wrong time. These traditional methods add considerably to the complexity of the system design, especially in intricate systems.

Because a State Machine model reflects sequence of operation over time, the model embedded in the controller matches the actual model the real world process follows. This model makes it possible to define the control system by describing the process.

Because the model matches the real world, program development and modification is always simpler and easier to understand. Program developers can more easily build advanced diagnostics for the process into the program because the control program is a precise model of the process and it is easy to detect when that normal behavior is not followed.

## A Collection of Tasks is a State Logic Program

Finite state machines or tasks define sequential operations. Processes though usually have more than one sequence of operations executing concurrently. State programs are usually a collection of Tasks matching the actual real physical Tasks that are inherently part of the process under control. The State Logic control program is a collection of Tasks which execute concurrently.

# Developing State Logic Programs with ECLiPS

Developing State Logic programs can be characterized as entering a description of your control system into a **template.** The template is the Finite State Machine model built into the State Engine in the control hardware. ECLiPS is a tool and framework for entering that description into the template. ECLiPS will provide all of the commands and tools you will need to "load" virtually any control application into the State Logic template. The primary element of the template's structure is the TASK. Each Task can be subdivided into an unlimited number of States. The I/O related activity and State change rules are described in each State with a collection of STATEMENTS. Statements are the ECLiPS command set you use to describe what you want to have happen at each State of each Task. In ECLiPS Statements are normal English words, phrases or sentences. An unlimited number of Statements can be used in any State.

Therefore, State Logic programs are a hierarchy of TASKS, subdivided by STATES, described by STATEMENTS.

TASK: Drill — ***TASK* NAME**

State: Drill_Advancing — ***STATE NAME***
  Turn Fwd_Solenoid on. After 3 seconds start Drill_Motor.
  When Fwd_Limit_Switch is tripped go to Retracting State.
  Go to Send_Message_1 if 17 seconds pass. — ***STATEMENT***

State: Retracting
  Actuate Rev_Solenoid. — ***STATEMENT***
  When Home_Switch is tripped go to the Increment_Counter State.

State: Increment_Counter
  Add 1 to Parts_Count.
  Write "Parts Count equals %Parts_Count" to operator_display.
  If Parts_Count is less than 24 go to the PowerUp State.
  If Parts_Count is 24 go to the Send_Message_2 State.

State: Send_Message_1
  Write "DRILL BIT DULL" to message_board, go to Retracting State.

State: Send_Message_2
  Send "RUN COMPLETED" to operator_display, go to New_Cycle State.

**TASK:** Setup_DISPLAY

State: Operator_Panel

**Figure 3-3. Sample Task with Some Elements Labeled**

## Tasks - Sequences of States

By design a machine or process is a collection of Tasks that operate concurrently. A car engine has an electrical system, a fuel system, a mechanical motion system, cooling system, exhaust system and a starting system that, while independent in action, must be coordinated in time for the engine to work. Similarly all industrial processes, machines and systems will contain several Tasks that are mutually exclusive in activity yet joined in time.

While the Tasks are independent in action they are naturally related in time, since all Tasks come to life at power up and stop with shutdown. The control system designer can divide the overall process into individual Tasks to exactly mirror the system.

The types of Tasks that may be created are unlimited. Typical Task types include; motion control tasks, mode control tasks, filling tasks, measuring tasks, shutdown tasks, data recording tasks, diagnostic tasks, alarm tasks, operator interface tasks and so on.

## States - The Building Blocks of a Task

Task: Mix_Station

State: PowerUp
If Can_At_Mix is on, write “Mixing Can” and go to Lower_Mixer.

**State: Lower_Mixer**
**Run Mixer_Down_Motor. When the Mixer_Down_Switch is tripped, then go to Mix_Chemicals State.**

State: Mix_Chemicals
Start the Mixer_Motor.
When 10 seconds have passed, go to Raise_Mixer.

State: Raise_Mixer
Run Mixer_Up_Motor until Mixer_Up_Switch is tripped,
then go to Mix_Complete.

State: Mix_Complete
When Can_At_Mix is off, go to PowerUp.

**Figure 3-4. Five State Task Example with a Single State Highlighted**

In the automobile engine example we said an engine is viewed as a collection of Tasks; Fuel System Task, Electrical System Task, Starting System Task and so on. Each of those Tasks is further described as a precise set of States through which that Task will pass while the engine operates.

The automobile engine’s Starting System Task has several possible States. For example we know there is a State in which the key is on, the engine is not running and the starter motor is not cranking the engine over. We know there must be another State in which the key is in another position, the engine is not yet running but the starter motor is cranking the engine over. There are also States in which the key is on, the engine is running and the starter motor is no longer cranking the engine.

Each Task is divided into States. The aggregate activity described by all of the States of a Task defines all the possible behavior of that Task under all conditions. A State defines the values for the outputs, sends messages, performs calculations, and assigns values to data variables. States also describe transitions to other States. Only one State is active and executed in a Task at any time. If two States need to be active at the same time then a concurrent Task is required.

Every Task must have at least one State. When the controller is powered up, the Task goes to this State, called the PowerUp State, which is the first State in the execution sequence of the Task. Thereafter, activity can move to any other State based on the Statements in the active State.

## Statements - The Command Set for State Descriptions

```
State: Raise_Mixer

   Write "Mixer Moving" to Operator_Panel.
   Turn on the Mixer_Up_Motor.  When the Mixer_Up_Switch is
   tripped, then go to Mix_Complete.
```

**Figure 3-5. Example ECLiPS State with One complete Statement Highlighted**

In the automobile engine Starting System example, we would find that to make a complete description of the Starting System Task, activity would have to be described in greater detail. We could break down each State into a set of Statements that completely described the full and possible ranges of activity of that State.

Let's give the name "Starting" to the State in which we are actually trying to make the engine start up and run on its own. In the "Starting" State we could make a Statement like; "When the ignition key is in position three, go to the Crank_Starter_Motor State.", representing one of the Statements that form a part of the complete description of all possible actions of the Starting State in the Starting System Task. If the car was equipped with an automatic transmission the Statement might need to read; "If the transmission is in neutral or park and the ignition key is in position three then go to the Crank_Starter_Moter State.".

The actions of a State are described with a Statement or a collection of Statements. In ECLiPS a Statement is a collection of Terms describing the desired actions for that State. Statements end with a period (.) and can be thought of as sentences, although punctuation and proper grammar are not required.

There are two types of Terms used in a Statement; Functional and Conditional.

Functional Terms perform a specific action, including turning on digital outputs, setting analog outputs to values, performing calculations, setting variables to values, transferring to another State or communicating with other devices.

Conditional Terms perform some decision making test which enables or prevents execution of the functional Term in the Statement. The conditions that can be checked for include digital point status, analog values, a read from a serial port, or status of any system variable, including State activity from other Tasks.

Functional and Conditional Terms are listed below using typical ECLiPS terminology.

| Functional Terms | Conditional Terms |
|---|---|
| Actuate, Start, Turn on | If |
| Go | when |
| Add, Subtract, Divide, etc. | Read |
| Make, Set | get |
| Write | |
| Start_PID, Stop_PID | |
| Suspend_Task, Resume_Task | |
| Perform | |

State: Wait_For_Command
Read Start_Command from Operator_Panel, then go to the Start_Process State.
If 20 seconds pass go to the Operator_Prompt State.

State: Operator_Prompt
**Write "PLEASE SELECT BATCH AND START PROCESS" to the Operator_Panel, then go to Wait_for_Command.**

State: Problem_Report
**Write "PROCESS SHUT DOWN BECAUSE MATERIAL IS TOO THICK".**
Go to PowerUp State when Reset_Button is pushed.

**Figure 3-6. Highlighted Communication Functions**

The Adatek controllers have two very powerful serial communication functions. These are a Read and a Write Term for the various serial ports.

The Write Term allows characters to be written to any of the serial ports in the controller. These can be connected to operator interface terminals or smart panels to present full screen displays or simple messages. These ports can also be connected to intelligent actuators or control devices, such as a robot controller, to provide set points and operating commands.

When a Read Term is encountered in the execution of a State, it is treated as a Conditional Term that isn't satisfied until characters are received from one of the serial ports. Once the complete message is received it places the characters in the designated variable for use by the rest of the program and then allows the active State to execute the next Statement.

The Read Term can be used to communicate to any serial input device. This would include operator interface devices such as terminals, smart panels, and personal computers. It would also include intelligent sensors such as weigh scales, and the various smart pressure and flow transmitters now sold by various manufactures.

Together the Read and Write Terms make communicating with the operator very powerful yet simple. It also makes it easy to communicate with intelligent sensors, controllers and other machines that populate the plant or factory.

Any one of the serial ports may be set up for the CCM2 communications protocol. Using this protocol enables the State Engine controller to be a slave on a CCM2 network. Typically the protocol is used to communicate with Graphical User Interface Software such as CIMPLICITY, Genesis, The FIX, INTOUCH, Factory Link, Screenware II, etc.

## Scan Overview

The State Engine which executes the control program continuously scans the inputs and the control program. These scans occur hundreds of times each second. Before each program scan, all of the inputs are scanned, so that each Statement of the program makes decisions based on the same input information.

During the scan of the program, the active State of each Task is scanned. Each Statement of a State is scanned in order from the first Statement to the last unless a GO Term is encountered. As soon as a Go is scanned, no more Terms in this State are scanned, and the scan moves to the active State of the next Task. Another State is scanned during the next scan sweep.

While scanning a Statement, the scan evaluates all conditional Terms before implementing the action described by the functional Terms. If any conditional Term is not satisfied or false, the scan of this Statement is stopped, the functional Terms are not implemented, and the scan resumes at the next Statement of the State.

The State Engine keeps a table of all digital outputs and flags which are set ON during the program scan. Only the outputs set ON by one of the functional Terms in one of the active States during the scan are set ON, all others are OFF. The real world outputs and flags are set ON at the end of the scan. Therefore, an output does not go OFF during the transition from one active State to the next when that output is set ON in both States.

This scan discussion is a general overview of the program and I/O parts of the scan. The reference section of this manual has a more detailed discussion of the State Engine scan procedure.

## Interaction Between Tasks

Returning to the automobile engine State Logic model, we can see that we could describe the entire range of actions of an automobile engine as a collection of Tasks. Further, we can identify the different States through which each individual Task could pass during operation. Further, we should be able to see how we could use Statements to describe all of the actions possible for each State and the input conditions that would dictate which of the possibilities would actually happen.

But the engine wouldn't work unless we synchronized the timing of Task's execution with one another. The Starting System can work perfectly but if the Fuel System Task doesn't provide a squirt of gas into the cylinder during the time the Starting System Task is in the Starting State the engine won't run. The same is true of the Electrical System Task, which needs to provide voltage to the spark plug at the right time in relation to the Fuel System Task, Mechanical System Task and Starting System Task if the engine is to start running.

All Tasks have two natural synchronization points, PowerUp and Shutdown. In between the Tasks will execute based on their own instructions and without regard to the other Tasks unless the program developer instructs the Task to do otherwise.

Joining the Tasks in time at various points in the operation of the process under control is not difficult. There are several techniques for accomplishing this coordination. A good working knowledge of how to implement Task interaction is important to the efficient development of State Logic control programs.

The following are a few examples of situations and techniques for controlling Task interaction.

1. Using a Variable to link Task Activities. One way to communicate between Tasks is by using a variable. Any Task has access to all variables even if the value for that variable is controlled by a different Task. An example of States in two Tasks using the same variable:

```
Task: Make_Parts
    State: Refill_Bin
    If the Parts_count is greater that 100 pieces go to the
    Refill_State. Otherwise go to the Grab_Base_Part State.
```

```
Task: Conveyor_Control
    State: Start_Station
        When Part_In_Place Switch is tripped, then Add 1 to Parts_Count
        and go to Start_Conveyor.
```

2. Changing the State of one Task from another Task. A common technique for implementing this type of Task interaction might be in connection with an Emergency Stop Button. Commonly, a designer may want every State of a Task or Tasks to take a specific action should an E-Stop Button be pushed. It would work perfectly well to specify the recognition of and reaction to the E-Stop button activation in every State, but this would be unnecessarily cumbersome.

A much more efficient method would be to create a separate E-Stop Task that forces a change of States in other Tasks when the E-Stop Button is activated. The State that the other Tasks would transition to by the by the E-Stop Task would contain a description of the desired response to the E-Stop button being pushed. This is how such a Task might look in ECLiPS.

```
Task: E-Stop

    State: Emergency
        If the E-Stop-Button is pushed put the Generating Task into the
        Safe State and the Switching Task into the Controlled_Stop
        State then go to the Wait_for_Reset State.
```

3. Using the Current State of a Task State as an Conditional Term. The current active State Status of any Task is a variable in ECLiPS and can be treated as an input condition to make a Conditional Term within a State of any other Task. An ECLiPS example follows:

```
Task: Start_Motors

State: Check_Conditions
    If the Conveyor Task is in the Running State then go to the
    Start_Main_Motors State.  Otherwise go to the Send_Message State.
```

4. Using internal flags to signal another Task. Internal flags are set and tested just as are digital outputs. One or several Tasks may set a flag for another set of Tasks to test, for example:

```
Task: Smoke_Alarm_Monitor

    State: PowerUp
        Turn on the Smoke_Alarm_OK flag.
        If the Dock_Detector is on or the Boiler_Detector is on, or
        the Transfer_Detector is on go to the Alarm State.
```

The Smoke_Alarm_OK flag is true until one of the detectors is activated and Alarm becomes the active State. Any other State may test this flag to instantly see whether there is a smoke alarm activated.

## Creating Process Diagnostics

One of the advantages of using the Adatek State engine approach to control is the ease with which on-line process diagnostics can be added to the control program. Because the control program describes the process, any aberrations to the normal process can be detected and a response pre-programmed.

## Creating Diagnostic Routines

The diagnostics can be added as Statements inside of States in Control Tasks, or whole new States within the Tasks, or as complete new Tasks.

If the desired response to an abnormal occurrence is simply a message or closing a digital output to turn on a light or sound an alarm, then that condition should be inserted as a Conditional Term in the appropriate Task.

```
If Tank_Pressure exceeds 90 psi, then write “OVER PRESSURE
CONDITION!” to the Operator and turn on the Alarm and go to the
Alarm_Light State.
```

If the occurrence of the condition needs to trigger a more elaborate response, or if it should alter the normal sequence of operation, then a Conditional Term should be inserted that is followed by a "go to" instruction that transfers the Task to a State where the diagnostic procedure takes over. If the occurrence can happen in multiple States, then a separate Task that checks for the occurrence and forces the control State into the diagnostic State may be the best way to perform the on line diagnostic.

Because the control program written in ECLiPS is self descriptive, and each State describes what should be happening and what should happen next, it is easy to insert diagnostics after the control program is finished.

In addition to the ability to add diagnostic logic with Statements, States and diagnostic Tasks, ECLiPS also contains several functions to help the user automate the process of adding them.

There are three primary techniques for adding diagnostic capability to a State Logic control program.

1. Each time a new State is added the user is given the opportunity to enter a maximum time for which that State can be active. The maximum time selected will be shown automatically in the program after the State name.
2. An "add diagnostics" choice is available from the menu. If selected the user will be given a choice of the type of diagnostic to add and a fill in the blanks screen. Once the screen is filled in, the Diagnostic State will be written automatically and inserted into the program.
3. Diagnostic logic can be created in the same fashion as control activities are accomplished, by using Task, States and Statements to describe the desired diagnostic activity.

Task: PINPOINT_FAILURE

State: Check_Pressures

If main_tank_pressure is less than 100 psi write “Pressure too low, check tank door seal” to operator_panel and go to the PowerUp State. If main_tank pressure is more than 250 psi go to the Issue_Warning State.

State: Issue Warning

Write “PRESSURE ABOVE NORMAL”. Go to the Pinpoint_Problem State.

State: Pinpoint_Problem

If Plant_Overview Task is in the Start-Up State and Pump_One is on, write “MAIN PUMP ON DURING START-UP - SHUTDOWN WILL BEGIN IN 30 SECONDS” to the Plant_Alert_Board and go to the ShutDown State. If the Normal_Run State of the Pressure_Control Task is active and the Relief_Valve is true write “Main Tank pressure relief valve is probably jammed” to Terminal_3.

State: ShutDown

When 30 seconds have passed, go to the Begin_Shutdown State.

**Figure 3-7. Example Diagnostic Task**

# Chapter 4

# Creating a State Logic Control Program

This section presents the fundamental concepts of how a control program can be built using ECLiPS. Every designer will develop his own style in using ECLiPS. ECLiPS is designed to support and even to encourage personal or corporate program development styles. Initially however, it is suggested that the following procedure be followed in creating your first control system program with ECLiPS. This procedure is split into two steps:

1. Outline the Application
2. Write the Program

# Outline the Application

In this step the control problem is analyzed using a top down design strategy where the components of the main problem are identified at the top level and then each of these components is broken down into its separate parts. This decomposition of the problem continues until the application is completely described. The State Logic Control model invites top down design because of the hierarchy of its elements, Tasks, States, and Statements as described in the previous section. There are several different formats to aid in the top down design process including structure charts and structured flow charts, but we use a simple outline approach.

## Identify the Tasks

The goal of this step is to identify the Tasks of the application. We start at the highest level, decomposing the problem into its general components. See the discussion on Tasks in the State Logic Control Theory section of this manual.

Think of the independent operations which must be accomplished to achieve goals of the application. The natural separations of activity often become Tasks.

The goal is to decompose the problem into parts that can be defined as sequences of I/O operation. Any cycles which repeat even with some variations are prime candidates to be Tasks. An important concept for identifying Tasks is that Tasks are a set of sequential operations. Events which occur in parallel or concurrently should be in separate Tasks.

These main sections of the outline should be general descriptive phrases such as:

| |
|---|
| Bore Cylinder |
| Load Boiler |
| Fill Vat |
| Retrieve Part |

At this stage the goal is to just describe the application not force some solution. Some of the independent Tasks are quite obvious, others which require interaction with other Tasks are more difficult to identify at first. This is usually a repetitive process where original efforts must be adjusted as the outline progresses. As with most activities, proficiency increases with the number of efforts.

## Identify The States

Once the Tasks are determined, then the States of each Task should be identified. The States describe the actual condition the outputs and responses to inputs at a certain point in the control process. The States form the control sequence and are really a picture of how this piece of the process (Task) should behave. See the discussion of States in the State Logic Control Theory section of this manual.

At this point in the design stage the goal is to determine that the correct action can be accomplished with the chosen Task architecture. Simply give each State a descriptive name fitting the major attribute of the activity that takes place when that State becomes active. Typical State names are:

```
Send Message
Add Water
Raise Drill
Start Motor
```

State names identify the general action of the State. The specific actions and the transitions are specified in the Statements.

## Identify the Statements

This level is the most specific level of the outline. Statements specify detailed actions which are to occur while this State is active, and the transitions to other States. See the discussion of Statements in the State Logic Control section of this manual.

The actions specified by Statements include digital outputs that are to be ON, changes in analog output values, changes in variable values, and messages to be sent. Examples of actions as specified in Statements are:

```
Turn ON Pump 5.
  Start Mixer Motor.
  Write "Operation complete" to Operator.
  Add 1 to Parts Count.
  Turn ON Forward Solenoid.
```

Statements also specify the transitions of a State. Both the condition for transition and the target State are identified. The status of digital inputs, values of analog inputs and variables, and elapsed time are used to specify conditions for a transition. State names specify the target State that becomes active when the condition is true. Typical transitions as they would appear in an outline are:

```
If Forward Limit Switch is ON, go to the Drill State.
  If Vat Temperature is less than 45 degrees go to Raise Temperature State.
  When 10 Seconds have Passed, go to Raise Mixer State.
  If Part in Place Switch is ON and Manual Switch is OFF, go to Move State.
```

Statements are often complete English sentences, since very specific operations are specified at this level of the outline. In fact, feel free to specify Statements in any comfortable format. Some additional examples combine the State actions with the transitions:

```
Run Mixer Motor. When 5 seconds have elapsed, then go Raise Mixer State.
  Write "Drill Bit is Dull" to operator, then go to Retract Drill State.
  Read Command from Operator, then go to Report State.
```

# Writing The Program

With this outline in place, the ECLiPS program is almost completely written. The finished program is very close to the outline.

There may be some changes to the outline because of some naming conventions for how Task, State, and some other names are entered into the program. ECLiPS can not provide for the full expressiveness of the English language so some of the sentence constructions may have to be changed, although many alternative structures and the ability to make custom changes to ECLiPS are provided. Also, the outline is in a general format with no specific reference to the actual I/O of the system so that the wording of the outline usually becomes more specific in the program.

To write the program the Tasks, States, and Statements of the outline are entered into the project using the ECLiPS editor which is active whenever ECLiPS is in Program Mode. Another part of creating the program is specifying I/O names and circuit configurations. Defining the I/O may be done before, after, or during the writing of the program.

## Using English Names in the ECLiPS Program

When you start a new program, ECLiPS asks for the name of the first Task. After the name is entered, ECLiPS starts the program for you by writing the Task keyword followed by a colon and the Task name. ECLiPS also writes the first State name, "PowerUp" into the program. Tasks, States, I/O points and variables can all be assigned English names. Names can be as brief and code like or as descriptive as you wish.

Clever, descriptive names that fit well to the primary attribute of that State activity is strongly encouraged. This will pay dividends in future program modifying, clear documentation and easier troubleshooting.

Further, good descriptive names will enhance the quality of the automatic diagnostics that can be created by linking Task, State and I/O names together for automatic diagnostic output information.

Each name can have up to a twenty characters. These characters may be letters, numbers, or the underscore character (_). Names must begin with a letter. The name must be a continuous string of characters, i.e., no spaces are allowed.

Because ECLiPS uses the space character as a way to tell where one word ends and the next begins, as we normally do in written English, a name can not contain a space. To construct a multiple word name for descriptive purposes the designer should use the underscore character (_) to separate words or use uppercase to start every new word.

| |
|---|
| Table_Movement |
| TableMovement |

## Naming Tasks

Task names are arbitrary. It is suggested that Task names be descriptive of the activity they represent. This descriptive use of names means clearer documentation and the ability to create automatic diagnostic output messages by combining Task, State and I/O names to make complete messages.

A Task may be added to the program by using the "Add a New Task" option from the Add Menu or just typing in the Task keyword followed by a colon and the Task name. Each Task is assigned a name as it is built and each Task must have a unique name. This name appears at the beginning of every Task. Every time the designer wants to refer to the Task using ECLiPS such as in writing other Task sequences, during debugging or during diagnostics development, the English Task name should be used.

## Naming States

Each Task contains one or more States. Similar to Tasks, a name is assigned to every State of the program either through the Add menu or directly into the program using the ECLiPS editor. Once assigned, these names are used when performing any functions associated with States while using ECLiPS.

Each Task always begins execution in the PowerUp State when the program starts. As a reminder as to which State will begin the sequence when power is applied to the controller, one State of every Task must be named PowerUp.

While every Task in a controller must have a unique name to differentiate it from the others, States in different Tasks may have the same name. All States within one common Task must have a unique name, but a State in one Task can be named the same as one in a different Task.

As with Tasks, names chosen for the States should be descriptive. By using combinations of words that describe something unique to the State, such as the action performed or function of the State, the program becomes self documenting. Using descriptive names makes it possible for people other than the original designer to use and modify a sequence at a later date with minimum learning time spent trying to understand the program.

## Naming I/O Circuits, Variables, and Internal Flags

Each input and output from and to the field enters and leaves the controller through some particular hardware module. A name is given to each of these I/O points. All references to I/O circuits use the assigned name.

Names are also used for variables and internal flags. All names must be unique. A variable must not have the same name as a State or a flag must not use the name of a Task. The only exception to this rule is that States in different Tasks may use duplicate names.

The English name should be descriptive and can be made up of several words attached by the underscore character. ECLiPS allows the user to define I/O points or other name other elements of the program at any time during the programming process.

## Statement Structures

This section describes how to express the Statements in an ECLiPS program.

```
State: Drill_Advancing

Turn Fwd_Solenoid on.  After 3 seconds pass, start Drill_Motor.
When Fwd_Limit_Switch is tripped go to the Retracting State.
If 17 seconds pass, go to the Send_Message State.
```

**Figure 4-1. Example ECLiPS State with Four Statements**

Most States consist of several Statements that describe what action is to happen while the Task is in that State, what conditions will cause a transfer, and to what State the Task transfers to. With ECLiPS these Statements are written in descriptive English generally but not necessarily consistent with the rules of English grammar. Statements are short sentences or phrases that describe the desired actions in a way that anyone can read and understand. A Statement always ends with a period just as a sentence does in English.

## Constructing Statements

There are two types of Terms in a Statement, functional indicating some action taken and conditional indicating some test for decision making.

After 3 seconds pass **start Drill_Motor.**
**Turn Main_Heater on** if Start_Switch is on.
**Write "Parts Run Complete" to User_Panel.**

**Figure 4-2. Statement examples with Functional Terms highlighted**

```
After 3 seconds pass start Drill_Motor.
Turn Main_Heater on if Start_Switch is on.
Open Vent when temperature is greater than 100 degrees.
```

**Figure 4-3. Statement example with Conditional Terms highlighted**

Functional Terms describe an action to perform when they are reached in the execution of a Task. Conditional Terms describe a condition that needs to be evaluated to decide whether the Functional Terms in the Statement should be executed at this time.

A Functional Term, such as "turn_on Motor_A" or "close the Red_Clamp", generally has a verb that describes the action such as, "turn_on", "close", plus a variable name or I/O name, such as "Motor_A", "Red_Clamp".

Terms are combined to form Statements. Most Statements will be a sentence or a phrase. A Statement may be entirely made up of a Functional Term such as "Turn_on the Automatic_Mode_Lite.". A Statement can also be a combination of Functional Terms such as "Turn_on the Automatic_Mode_Lite and the Main_Conveyor.". Often a Statement is a combination of a Conditional Term and a Functional Term such as "If Motor_A is on turn_on the Automatic_Mode_Lite and start Main_Conveyor.".

Every Statement must always have at least one Functional Term. A Statement can contain more than one Conditional Term, or a Conditional Term that is a combination of conditions, such as “If Motor_A is on and the Red_Clamp is closed” or “If Motor_A and Main_Conveyor is on”. There may be many Functional and Conditional Terms in a Statement.

## Using Keywords, Synonyms and Filler Words

Keywords are the words in a Statement that ECLiPS recognizes as instructions to perform some function. Keyword can be words that cause an action, such as the word “actuate” when applied to a contact output. Or they can cause a conditional comparison such as the word “if”, or be part of the comparison such as the symbol “ > ”.

ECLiPS comes with default keywords assigned. Some of these keywords also have synonyms defined. Using a synonym in the program is the same as using a keyword. A detailed list of all the keywords are given in the reference section and described in detail.

ECLiPS also comes with several filler words such as “the” or “a” defined. Filler words have no meaning to the control program. The sole purpose of filler words is to make program Statements more readable and understandable. The user can place filler words anywhere in the Statement. Commas and other punctuation may also be used for clarity without effecting program execution. The only punctuation which has meaning is the period (.) and the exclamation mark (!). The exclamation mark is used to document or add comments to the control program.

Keywords are the vocabulary of ECLiPS and together with filler words make it possible to easily write understandable descriptions that are the control program. This vocabulary may be changed to suit any desired convention. All of the keywords, synonyms, and filler words may be changed. ECLiPS can therefore be configured so that the program is written in a foreign language.

A menu based window function allows the user to make these assignments at any time during an ECLiPS programming session. In addition, another menu based window function allows the user to see a list of all synonyms previously assigned and to select one to enter into the program.

Use the flexibility to create a language that fits the terminology of the industry, or plant, etc. where ECLiPS is to be used. The written ECLiPS programs become even clearer to all involved with operating and maintaining the plant as they use the English names for the process points and the local terminology for the actions and descriptions.

## Using Variables

Statements manipulate variables in addition to controlling the I/O. At times the application requires responses to values other than those represented by real field sensors. Examples of this might be the number of parts built during a shift, the flow through a pipe calculated based upon the pressure drop across the pipe, the style of part being built this production run, etc.

Items stored in variables are the results of calculations, totals that are being accumulated over time, something that must be remembered from one time period to the next, and constants that may be changed or tuned. Each variable is assigned a name during program development.

Once created each variable is available to be shared between Tasks and States within Tasks. One State may assign a value to the variable and another State (or the same State at a later time period) uses the value of the variable in making a decision. Once a variable is assigned a value, that variable maintains that value until a program Statement assigns a new value to that variable.

Variables may be configured to save their values when the program is halted. The values in the State Engine cannot be saved over a power cycle since computers normally do not have non-volatile memory. If not configured to save values, a variable is always set to zero when the program starts running. Variables are always initialized to zero after a power cycle.

When using ECLiPS to write programs or monitor running controllers, or generate diagnostics, the user needs to only refer to the variable by its English name. Remember that choosing descriptive names for variables helps to make the program self documenting.

The different variable types are listed below:

## Integer Variables

This type of variable represents a whole number from +32767 to -32768. Integer variables have many uses including counts, menu choices, and item quantities.

Integer variables can also be used as logical variables, or variables that have only two possible values, either 1 or 0. Variables used in this manner can be thought of as true or false, on or off, etc. Using integer variables in this manner differs from using flags, since flags are ON only when a State turning them On is active. On the other hand a variable maintains its value independent of which State is active.

## Internal Flag

Internal flags are variables that act like digital outputs, but do not produce any physical output from the controller. An internal flag can be set true by a State in one Task and then checked by a State in another Task. These flags can be used to coordinate the actions of different independent Tasks.

An internal flag is like a digital output in that if an active State is not setting it true the controller will automatically turn it OFF.

## Floating Point Variable

This variable type is used to store numbers that are not whole numbers or numbers outside the range of integer variables. When variables are needed with a math function, generally it should be a real variable.

## String Variable

This variable type stores a collection or “string” of characters. These characters can be any alpha numeric or control character represented by an ASCII code. This type of variable is a little more complicated and is used mainly in accepting inputs or creating outputs to serial communication devices.

### Character Variable

Character variables store one single ASCII character. This type of variable is especially useful for operator interfaces when the operator must enter a single character.

### Time Variables

The time variables are used to view or change the values of the Real Time Clock. The time variables are second, minute, hour, day, day of the week, and month. Use these variables to set the clock or to check the current time.

## Program Scan

The State Engine operating system is a scanning system. The scan cycle starts at the start of the program, scanning the active State of every Task. During program execution there is always one and only one State active in each Task. The operating system completes a scan of the program hundreds of times every second.

During the scan of the active State of a Task, each Statement of the State is scanned in the order that it appears. Keep in mind that a Statement is a series of Terms terminated by a period (.).

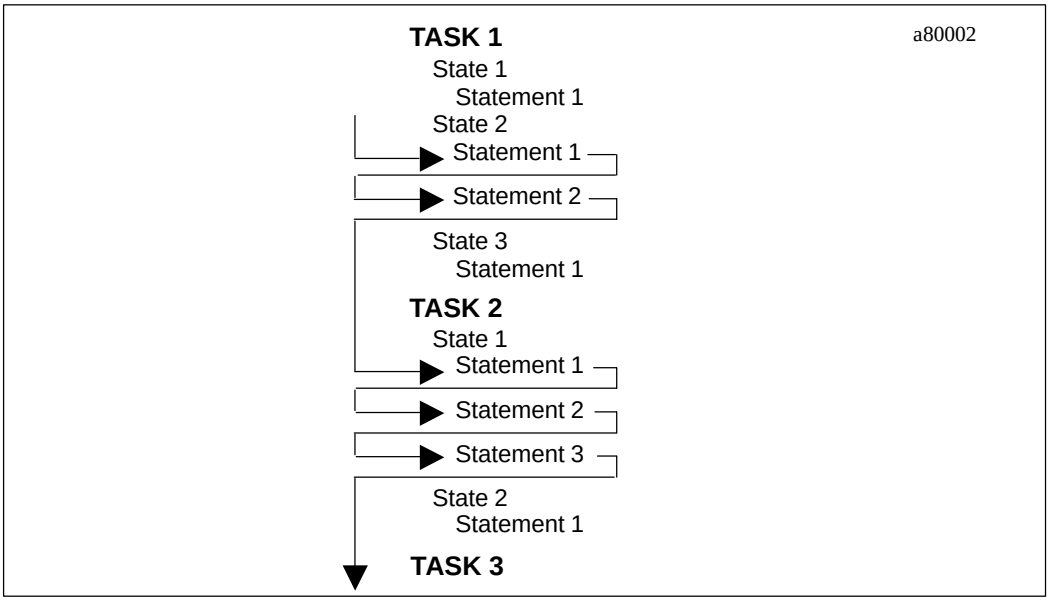


**Figure 4-4. Program Scan**

## Program Scan

The actions specified by Functional Terms are executed when the Functional Term is scanned. Each Statement must have at least one Functional Term, Conditional Terms are optional. If there are no Conditional Terms in a Statement, the Functional Terms are always executed during each scan. When Conditional Terms accompany Functional Terms in a Statement, the Functional Term is executed when all of the Conditional Terms are satisfied. There are four types of conditional Terms (see the reference section).

Conditional Terms are satisfied as follows:

1. Read - When valid data is received at the appropriate channel.
2. If - When the conditional expression is TRUE.

To understand how Statements are scanned, assume that the Statement Conditional Terms precede the Functional Terms and that the scan proceeds from left to right.

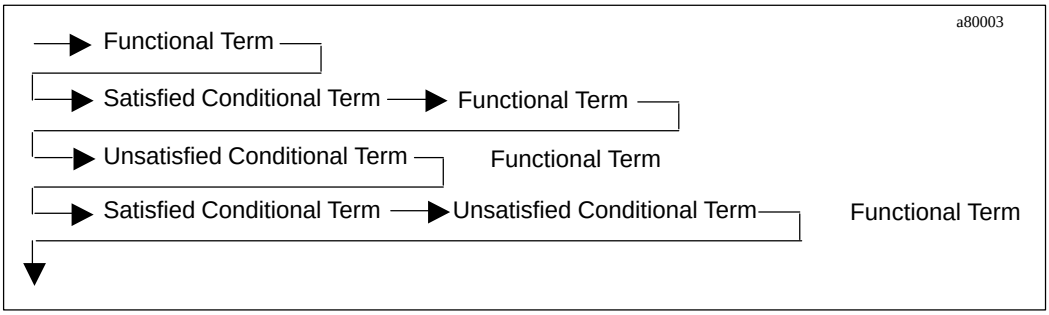


**Figure 4-5. Statement Scan**

The Statements of a State are executed in the order that they are written into the program. Functional Terms of Statements with no Conditional Terms are always executed. Conditional Terms in Statements control whether or not the Functional Terms in those Statements are executed. If all of the Conditional Terms are satisfied, the Functional Terms are executed. If any of the Conditional Terms are not satisfied and Conditional Terms, the Functional Terms are not executed. For simplicity this rule assumes that the Conditional Terms are ANDed together. See the reference section about combining Conditional Terms using the AND and OR logical keywords.

The Statements are executed one at a time. In this manner every Statement of the active State is evaluated.

There are two types of Functional Terms that can prevent the execution of the rest of the Statements in a State. One is the Halt command which stops program execution. The other is the “go to ...” command, which immediately causes another State to become the active State. No Terms in a State are scanned after a GO is scanned in that State.

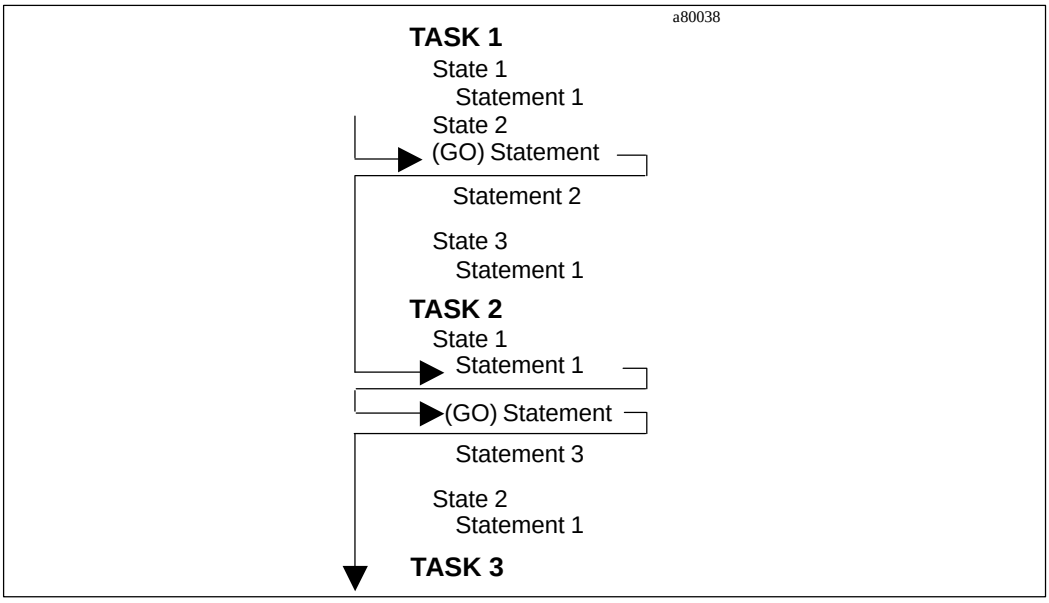


**Figure 4-6. Program Scan with GO Terms**

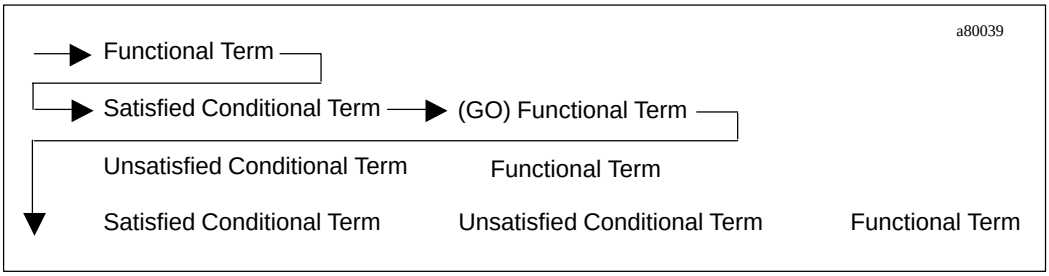


**Figure 4-7. Statement Scan with GO**

```
If Start_Pushbutton is pushed, go to Start_Up State.
Write “Press Start Push Button Now!!”.
If 30 seconds have passed, go to the Restart_Buzzer State.
```

This series of Statements causes the Start_Up State to become the active State when the input represented by Start_Pushbutton name is true. When GO Term is scanned, all Terms or Statements following this Term are not executed and at the next controller cycle, the scan of this Task starts at the first Statement of the Start_Up State.

During the program scan any changes to variables are made immediately. Therefore, a variable change in one Task is visible by the rest of the program during the same scan. On the other hand, digital I/O and Flag and analog values are made at the end of the scan. Therefore, if one Task makes a change to the condition of a digital output or Flag, the condition cannot be tested by another Task until the next scan through the program.

# Hints for Creating ECLiPS Programs

## Outputs are OFF by Default

One of the key features of the State Logic model is that the discrete outputs are OFF by default (their normal condition). Outputs are only ON if they are being turned ON by a program Statement in an active State.

This arrangement enables the State Logic model to have a one-to-one correspondence between the control program and the real world. States in the program exactly reflect the States of the machine being controlled. There are other direct benefits of the outputs being OFF by default.

While writing the State Logic program, the programmer need not be concerned with turning any outputs OFF. Having to be concerned with turning outputs OFF adds much complexity not to mention a great deal of logic to a control program.

Another feature of this arrangement appears when troubleshooting or debugging an executing program. To see which outputs are ON at any point in time, one only need check the State definitions of the active States of each Task. Any outputs set ON in these States are ON and all others are OFF.

Outputs are turned on by using the keyword Start (or any of the synonyms such as Turn_On, Energize, Actuate, etc.). Outputs are turned off when another State that does NOT turn on the output becomes active. An example is listed below:

```
Task: Drill_Press_1
   State: PowerUp
      If Clamp1 and part_in_place are true, go to Advancing.

   State: Advancing
      Turn_On Forward_Motor. When Drill1_Forward_LS is true then
      Go to Retracting.

   State: Retracting
      Turn_On Reverse_Motor. When Drill1_Home_LS is true then
      Go to the Counter State.
```

In the Advancing State the Forward_Motor is turned on. When the Drill1_Forward_LS digital input is true the machine will go to the retracting State. The Forward_Motor output will then be turned off because it was NOT turned on during this State.

The operating system assumes that if an output is not actuated during an active State that the output is OFF. When actuating an output remember to continue to actuate (or turn_on or energize) that output in all successive States that also require that output to be ON.

### Note

An output stays ON during the time when the operating system goes from one State definition to another, since State transitions do not take any time.

The following example demonstrates how to keep an output ON for successive State definitions:

```
Task:  Fill
        State:  PowerUp
                Energize Fill_A.
                Write “Filling tank with liquid A”
                Go to the Weight State.
        State:  Weight
                Energize Fill_A.
                If Weight_Input > 20 lbs,
                Go to  Fill_B State.
```

If there are several States where the same outputs are turned ON again and again, then probably the program should be rewritten and another Task added to control the outputs that are ON in several successive States. In the following example the output Run_Light would be on at all times unless the E_Stop button had been pressed. See Task Design in following section for more on this subject.

```
Task: Output_Always_On
        State: PowerUp
                if Emergency_Stop is on go to E_Stop.
                Turn on Run_Light.
        State: E_Stop
                If Reset_Buttton is pressed go PowerUp.
                Put Other_Operations Task Into Emergency State.
Task:  Other_Operations
        State: PowerUp.
                Turn on First_Operation.
                if  2 seconds have passed go Next_Step.
        State: Next_Step
                Turn on Second_Operation
    .           if 2 seconds have passed go PowerUp.
        State: Emergency
```

**Figure 4-8. Using multiple to Tasks to keep an Output ON**

## Task Design

Tasks should be designed to control operations that are executed in parallel or at the same time as other operations. This might be a motion operation that happens in parallel with operator interface activity.

An emergency stop push button is an example of a function that should be placed in its own Task. The Task can be called ESTOP for example and its only function is to monitor the emergency stop button and coordinate the activity of the other Tasks.

```
Task: ESTOP
        State: PowerUp
              Let Reset = 0.
              If the emergency_stop button is pressed,
              Go to the Shut_down State.
        State: Shut_down
              Make Fill_Can Task = Shut_Down State.
              Make Conveyor Task = Shut_Down State.
              Make Temp_Control Task = Shut_Down State.
              Go to the Wait_Reset State.
        State: Wait_Reset
              If the Reset button is pressed Let Reset = 1.
              Go to the PowerUp State.
```

This example uses one Task to determine if an emergency stop button has been pressed and then forces the other Tasks to go to their own shut down States. Notice that States in different Tasks can have the same name. The integer variable Reset is used to communicate to the other Tasks when the Reset button has been pressed.

- An indication that another Task should be created, is that a program segment requires an output to be turned ON in several States or an input is repeatedly monitored in several States. This type of structure indicates that more than one activity is being controlled by one Task, and a new Task should be used so that Tasks are used to control only one activity at a time.
- When Tasks are used for more than one activity, the complexity of the programming increases. Therefore, inordinate program complexity is another clue that there are too many activities being controlled by one Task.

## Write Term Considerations

The write term is interpreted in a somewhat different manner from the rest of the terms. Most terms are executed every time they are scanned. Some terms may not be scanned if preceding conditional terms are not true, but the write term executes only once for the entire time that the state remains active. As soon as there is a state change, the write term again will write its message when scanned.

```
State: OverTempAlarm
    Write “Over Temperature Alarm! Press reset switch” to OperatorStation.
    If ResetSwitch is pressed, go to PowerUp State.
    Wait 10 seconds, then go to OverTempAlarm State.
```

The previous program segment writes the alarm message once every ten seconds, until the reset switch is pressed.

## Calculations and a Scanning Operating System

Because the State Engine operating system is a scanning system, each Statement of the active State of each Task is scanned many times every second. Because of this scanning design, care must be taken when using mathematical operations.

The make structure is executed each time it is scanned. If there is a mathematical operation in the Make, that operation is performed each scan. This may result in unexpected values generated by the Make.

```
State:  Check_For_Part
    Make Parts_Count = Parts_Count + 1.
    If Photo_sensor is ON, then go to the PowerUp State.
```

The State above will cause the variable Parts_Count to be incremented every scan that the State is active and the Photo_Sensor is not ON. Instead of incrementing the count by one, it may be incremented by several hundred.

The correct way to construct this counter is to put it after the conditional so that it is executed only once before the next State becomes active.

```
State:  Check_For_Part
    If Photo_Sensor is ON,  then Make Parts_Count = Parts_Count + 1 and go
  to PowerUp State.
```

## Read Term Considerations

A Read Term is the READ keyword followed by a variable. This Term causes the processor to Read an input from the keyboard or from a SCM port and store the input into the variable that follows READ. The Read Term is a conditional Term and must be followed by a GO Term. The Read is satisfied when valid input to the variable is completed.

Input to the variable is completed when valid data for the variable is received at the specified communications port. Any of the variables may be used with the Read Term, and the type of data received must match the variable type to be valid. If the data type does not match the variable type, the data received is ignored and the Read Term continues to wait for valid input.

If a character variable is used with READ, the first character received at the port is stored in the variable and the READ conditional is immediately satisfied causing the following GO Term to be executed. All other variable types used with READ, wait for an End Of Message Character to be received before the input is satisfied. The default End Of Message Character is a Carriage Return, so that normally the READ is satisfied when the <Enter> key is pressed. The End OF Message Character can be changed by using the Set_CommPort keyword.

It is possible to have two Read Terms for the same port to be active at the same time. This arrangement can only happen if the Read terms are in separate Tasks. When two Read terms are active at the same time, it cannot be predicted which one may receive the next input from the port.

It is a good practice to place all read terms for the same port in the same Task, so that only one Read is active at a time.

## Timer Considerations

All timers in ECLiPS monitor the time that the current State has been active. This method of establishing a timer applies to the IF, FOR, and WAIT conditional terms.

Another way to think of the Wait Statement is as a conditional term that says "On the condition that this State has been active for __ seconds ...". If the State has been active for the amount of time specified the condition will be seen as a satisfied condition. An example follows:

```
Task: Cook
        State: PowerUp
                Go to the Start_Cooking State.
        State: Start_Cooking
                Actuate Heater.
                Wait 10 seconds, go to the mixing State.
```

A logical ERROR in using a timer is demonstrated below.

```
Task: Drill
    State: PowerUp
        Go to the Punch_Down State.

    State: Punch_Down
        Energize Punch_Down_Output.
        When Punch_Down_LS is true and if 3 seconds have passed,
        Go to the Punch_Up State.
```

If it takes 3 or more seconds between the time the Punch_Down_Output is energized and when the punch down limit switch is met, the Wait timer is immediately satisfied. If the desired action is to wait 3 seconds after the Punch_Down_LS is true, then another State can be added as follows:

```
Task: Drill
    State: PowerUp
        Go to the Punch_Down State.

    State: Punch_Down
        Energize Punch_Down_Output.
        When Punch_Down_LS is true  go to the Punch_Wait State.
    State: Punch_Wait
        Energize Punch_Down_Output.
        Wait 3 seconds, go to the Punch_Up State.
```

## Documentation Hints

This section offers advice on how to more effectively use the ECLiPS documentation features.

### Descriptive Names

When naming Tasks, States, I/O and other data types it is extremely useful to use descriptive names. This will prove to be very helpful when debugging the program and when troubleshooting the production machine. The following program provides an example of descriptive name use.

```
Task: Fill_Station
    State: PowerUp
        If Can_At_Fill is on, go to the Pour_Chem_1 State.

    State: Pour_Chem_1
        Turn on Chem_Valve_1.
        When Fill_Weight_Input is above 20, go to Pour_Chem_2.

    State: Pour_Chem_2
        Open Chem_Valve_2 until Fill_Weight_Input is above 30 lbs,
        then go to Wait_for_Can_Removal.

    State: Wait_for_Can_Removal
        When Can_At_Fill is OFF, go to PowerUp.
```

This program is easy to understand because of the names chosen for the Task, State and I/O channel names selected.

### Underscores

The use of underscores is also helpful when naming Tasks, States, and I/O. The previous example also demonstrates the use of the underscore to help clarify the name of an I/O point. The underscore is necessary for the compiler's interpretation of the English text. A word is always considered complete when a space is found after text. Therefore to separate letters in one word (like an I/O point name) the underscore acts as a space without causing the compiler to see the descriptive name as two individual words.

Another popular way to separate words is to capitalize each word but use no spaces.

| | | |
|---|---|---|
| Power_Up | or | PowerUp |
| Can _At_Fill | or | CanAtFill |
| Fill_Weight_Input | or | FillWeightInput |
| Punch_Down | or | PunchDown |

In either case it is best to be consistent throughout the project.

## Programming Conventions

It is also suggested that the company using ECLiPS decide on a few programming conventions to increase the standardized look of all of the programs created in one company. Some suggested conventions include:

1. The default keywords should be selected and used throughout the program.
2. Task and State names are typed in all upper case.
3. Standardize abbreviations such as LS for limit switch, LT for light, etc.

### Comments

Comments should be used in the program to clarify confusing or complex logic. To insert a comment in the program simple type an exclamation mark (!) followed by the comment. ECLiPS ignores any text following the exclamation mark on that line. Comments may be inserted after any program lines or take up an entire program line.

## Scan Time Considerations

This section presents a few ways to optimize program scan time. It is easy to forget that this system is a scanning system and that the current State of each Task is scanned repeatedly. A common mistake is to remain is a State that continuously performs some redundant operations such as initializing a variable value. Such operations need only be performed once then proceed to another State. Remember each time the operation is performed, scan time is used.

Another technique that can save scan time is to locate variables and I/O points at the lowest number location available. The State Engine accesses all memory locations up to the largest defined for that type in the program. For example, if the largest analog point is defined as %AI 900, all of the data from the start of the AIs up to 900 is read by the State Engine. If there were only one AI in this system, designating it as %AI 900 waists the scan time necessary to read in all of the unused AIs.

Another scan time saving technique is to split up large States which repeatedly check many items. If the items do not need to be checked every scan, splitting a large number of checks into several successive scans can reduce the scan time.

# Chapter 5 ECLiPS Programming Features

This chapter begins a series of chapters devoted to creating the State Logic program. This chapter describes the features that are used to create and modify State Logic programs. It also discusses the features used to convert and download a State Logic program to the State Engine.

A different chapter, devoted to ON - LINE features, discusses how the State Logic Programs can be used to control the State Engine.

This chapter highlights the following features:

| | |
|---|---|
| State Logic Word Processor | Features to create the program text |
| Project Management | Features to manage project files |
| Variable Name Definitions | Features used to define variable names |
| On - Line Program Changes | Changing the Program without halting |

# State Logic Word Processor

State Logic Programs are created in Program Mode. The State Logic Program is written in English using the State Logic Word Processor. There are a number of features in the ECLiPS State Logic Word Processor that facilitate the creation of State Logic Program. This section highlights the features available.

In ECLiPS the top banner always shows the name of the project. The bottom of the screen shows what key functions are currently available. Pressing the <F3> function key brings up the Program Mode Menu. The Program Mode Menu provides access to all of the features in Program Mode. There are several Hot Keys that can be used to short cut the menu steps.

Help is available in Program Mode by pressing <F1>. The Help system provides help that is in context with the operation currently being performed. Pressing <F1> twice brings up a list of all of the Hot Keys available.

## Creating Program Text - Overview

There are three ways that text can be entered into a State Logic program. The standard way that most text is entered into a State Logic Program is by typing data in. In cases where ECLiPS requires text with more structure, the ADD functions can be used. The LIST function on the Program Mode menu offers a third way to enter text.

### Add Functions

ECLiPS uses the Add functions to assist the State Logic programmer. These add functions use forms to create program code that requires more structure than normal English. The forms ask for the required data and then place the data into the program in the required structure. Some features available through add functions; Variable range checking diagnostic, User input menus, VME reads and writes, etc.

### List Functions

Choosing the list function displays a list of possible data types and information that can be listed. Selecting a data type lists all the variables of that data type. Placing the cursor on top of an entry and pressing return places that variable name into the program.

Example: The name for the system variable for the time is needed. Selecting the List option and the Reserved System Variable option lists the following variable names:

```
Day
Day_of_week
Hour
Minute
Month
Second
Time
```

Placing the cursor on the word time and pressing return enters the word 'time' into the State Logic Program.

## Text Functions - Block Functions

ECLiPS has several keys that can be used to manipulate blocks of text. These functions can be accessed through menus, or through the use of hot keys.

Block functions can be accessed by selecting TEXT from the Program Mode Menu or by using the Hot Key <F8>. The cursor is moved to select the text to be include in the block, press <enter> to select the text. The text that is selected is highlighted. A menu then gives the options; Copy, Move, Remove, or Perform multiple copies of the selected text. For copy, move, or perform multiple copies, move the cursor to the new text location and press <enter>.

The block functions make use of a buffer. Selected text (highlighted) is copied or moved to the buffer. The contents of the buffer can be placed in the program at any time by pressing <Ctrl + U>.

For example, press <F8> to select some text then press <enter> and <R> to remove the text, that text is still accessible by pressing <Ctrl + U>. That text continues to be in the buffer (therefore accessible), until more text is added to the buffer.

### The Hot Keys for Text Blocks are:

| | |
|---|---|
| <F8> | To select text press <enter> then use menu selections |
| <Ctrl + F8> | To select text and place it in buffer for remove |
| <Shift + F8> | To select text and place it in buffer for moves |
| <Alt + F8> | To select text and copy it to buffer. |
| <Del> | Deletes any highlighted text to buffer |

## Find Functions - Search and Replace

ECLiPS is equipped with search and replace capabilities. These functions can be accessed through the Program Mode Menu under FIND or through a series of Hot Key sequences.

The FIND feature searches for any specified text string. In addition, the FIND feature can be used to search for a text string and replace it with a different text string. It can also be used to move to different tasks, and to find the location of the last error message.

The hot keys that can be used to access the FIND functions are:

| | |
|---|---|
| <Alt + F5> | Search for a text string and replace it with specified text. |
| <F5> | Search for a text string |
| <Ctrl + F5> | Looks for each occurrence of a specified text string and gives the option to replace the string or to continue searching. <Esc> stops search. |
| <Shift + F5> | Finds every occurrence of a text string and replaces it with a different string. This command works on text in all of the task groups of a project. |
| <F7> | Looks for Another task, you must know the task name. |
| <Alt + F7> | Goes to the position of the Last Project Error |

### Cursor Control Keys - Hot Keys

ECLiPS uses a number of key sequences to facilitate the movement of the cursor through the document. These key sequences are referred to as hot keys. The following hot keys can be used to control the cursor:

| | |
|---|---|
| <Ctrl + Home> | Moves the cursor to the top of the program |
| <Ctrl + End> | Moves the cursor to the bottom of the program |
| <Home> | Moves the cursor to the start of the line |
| <End> | Moves the cursor to the end of the line |
| <Del> | Deletes single letter or highlighted block (see Text) |
| <Ctrl + Y> | Deletes the line the cursor is on. Text is lost. |
| <Ctrl + Left> | The Left arrow key with Ctrl moves cursor one word left |
| <Ctrl + Right> | The Right arrow with Ctrl moves cursor one word right |
| <PgDown> | Moves cursor down one page |
| <PgUp> | Moves cursor up one page |
| <Alt + F1> | Toggles between insert text and over type text |

# Project Management

ECLiPS has a number of features that are designed to assist in the creation, manipulation, and maintenance of the project files. These features are accessed through the PROJECT selection of the Program Menu (F3). Many of the features can also be accessed with the use of Hot Keys.

## File Management

The following features can be used to manage the project files.

**Retrieve:** Hot key <Ctrl + F2> Selecting retrieve from the menu displays a list of projects in the current directory. Highlight the project of choice and press return, and that project becomes the active project.

**Save:** Hot Key <F2> Saves and backs up the project

**Copy:** Copies the current project to a new directory or project name.

**New Path/Drive:** Changes the current directory

**Make a New Project:** Saves the current project and creates a new project

**Delete a Project from the Disk:** Deletes a project

**Import:** The Import function allows the combination of sections of other projects with the current file being used. There are several options under the import file menu choice.

The options, Import Merge Project Sections and Import English Text, are different in how they operate. When using Import English Text, the text file from the selected project is brought in as a new task group to the current project. ECLiPS does not look at variable names or Task and State names. Using “Import Merge Project sections” causes ECLiPS to import the text and the variable names. It also compares the import project with the current project to insure that there are no overlapping variable names, variable assignments, or task names.

## Documentation - Print Function

ECLiPS is equipped with a project documentation tool. Selecting the PROJECT "Print" function brings up a form to be filled out. The Print function has the following operations:

| | |
|---|---|
| Header | Text Printed at the Top of Each Page |
| Footer | Text Printed at the Bottom of Each Page |
| Output To | Any entry besides **Printer** directs documentation to a file with that name.- **Printer** entry sends output to parallel port printer |
| Auto Print | When entry is 'Y' this print setup is executed whenever the program is downloaded to the State Engine |
| Number of Copies | Number of copies to print |
| Lines per Page | Number of lines printed per page - some printers may need setting less than the 66 default |
| English Code | Prints the State Logic Program by Task Groups |
| I/OMap | Prints a list of all named I/O Points arranged in numeric order according to type |
| Data List | Prints a list of all variables and I/O points in alphabetical order according to type |
| Task / State List | Prints a list of Tasks followed by the State in each Task. The result is a useful outline of the entire project |
| Cross Reference List | Prints an alphabetical list of every named I/O point and variable and every Task and State where it appears in the program. This option may take a long time to execute. |
| CCM Protocol Listing | Prints the number and type to be used when communicating with the CCM protocol - see the SCM chapter |
| PID Loop Info | Prints all of the PID parameter initial value settings as specified in the initialization form |
| Keywords, Fillers, PID Parameters | Prints the current keyword and filler word definitions in addition to the PID parameter keywords |
| Last Uploaded Trace | Prints the Trace listing that was last viewed in ECLiPS or OnTOP |

## Error Checks, Translate, and Download Projects

The process of converting the English program text into a format that is executable by the State Engine is called translation. During the translation process the program is also checked for any errors. There are two options on the PROJECT menu to translate and error check the project: "Translate and Download Project to the Controller" and "Error Check and Translate the Current Project". The only difference between the two options is that the first also downloads the project to the State Engine when the translation is complete. ECLiPS saves the project to disk automatically before each translation. At the end of a successful translation a screen of statistics is displayed.

## Task Groups

Task Groups are simply a collection of Tasks. Using Task Groups is a way of breaking a large project down into smaller and more manageable sizes. When the "Make a New Task Group" option is selected, ECLiPS creates a new text file that is a part of the overall project. When a New Task Group is selected it also adds the TASK GROUP option to the Program Mode Menu.

The Task Group menu allows the Task Groups order to be changed. There are times although rare where the order of Tasks makes a difference in program execution. The Tasks at the beginning of the program are executed first. The Task Group menu allows the order of the Task Groups to be changed.

Instances where the Task order is important are when using variable or current State values in conditional terms. Variable assignments and State changes are visible to the rest of the program immediately. Discrete value changes are not apparent until the next program scan cycle.

Variable and I/O names and values are shared across the different Task Groups, and Tasks in one Task Group can control Tasks in another Task group. The following list displays options on the Task Group menu.

- Remove a Task Group
- Make a New Task Group
- Give the Current Task Group a New Name
- Change the Order of the Task Groups
- Join Two Task Groups Together
- Switch to Another Task Group (Hot Key <Ctrl + G>).

## File Types Created by ECLiPS

ECLiPS uses the concept of projects to keep work organized. A project is considered to be all of the files that are used in association with one application program. ECLiPS splits each project up into a variety of different files. All of the files in the project have the same root name with different extensions.

If the name of the project is Car_Wash, some of the files created by ECLiPS follow:

| | |
|---|---|
| Car_Wash.**PRJ** | Data about project; variable and state names, etc. |
| Car_Wash.**TG0** | The English text of the first task group in the project. |
| Car_Wash.**TG1** | Optional: The English text of the second task group. |
| Car_Wash.**TG2** | Optional: The English text of the third task group. |
| Car_Wash.**DBG** | Data about Debug operations, Force and monitor tables, etc. |
| Car_Wash.**PRT** | Optional: Logicmaster Configuration used in ECLiPS |
| Car_Wash.**PSM** | The translated program that is loaded to the State Engine |
| Car_Wash.**TRC** | Optional: This data is used by the trace function in Debug |
| Car_Wash.**PRB** | Backup of Car_Wash.PRJ generated automatically |
| Car_Wash.**BK0** | Backup of Car_Wash.PRJ generated automatically |
| Car_Wash.**BK1** | Backup of Car_Wash.PRJ generated automatically |
| Car_Wash.**BK2** | Backup of Car_Wash.PRJ generated automatically |

# Naming Variables

State Logic programs refer to variables and Digital I/O by names. Defining a variable requires; a name, a variable type (Analog Output, Integer, String) and the State Engine memory location (%AQ2, %R0001, %R1001 for example). There are three ways that variable name definitions can be made, viewed, and adjusted. The DEFINE menu, LIST menu, and DEFINE "System Configuration" all allow the user to work with the variable name definitions.

## Define Current Word - Search for Undefined Words

In the DEFINE menu, the "Define the Current Word" (Hot Key <F4>) option looks for the definition of the word that the cursor is currently on or closest to. If the word is defined, ECLiPS returns the name, data type, and memory location. If the word is undefined, ECLiPS goes to the Define Variable Menu where the word can be defined.

The Define Variable Menu provides a choice of different data types. The word can be left undefined or edited if the word is misspelled. Depending on the data type chosen, ECLiPS supplies a form that facilitates the definition of the variable.

Another option in the DEFINE menu is the "Define all Undefined words in the Text" option (Hot Key <Alt + F4>). This option starts at the beginning of the project looking for variable names that have not been defined. If ECLiPS finds an undefined word, the Define Variable Menu is displayed. After the variable has been defined, ECLiPS continues to search for the next undefined word.

## List - Variable Type

The LIST option from the Program menu allows the user to view, modify, and make definitions to variable names. Selecting LIST from the program menu gives a list of data types that can be listed. After selecting a data type ECLiPS lists all of the variable names of that type that have been defined.

Variable definition can be modified by positioning the cursor on top of the variable name and pressing the right arrow key. The variable definition can be deleted by pressing the delete key when the cursor is on the variable name.

Pressing the insert key when listing a data type, allows the addition of another variable name to that type of data. Example: while listing the Digital I/O variables, to add another %I variable named Extra_Input, press insert and add that variable name and fill out the data entry form.

## The System Configuration Option

After the PLC configuration is completed using Logicmaster, a visual representation of the racks and modules is available by first selecting the "Retrieve Logicmaster Configuration Data" from the DEFINE menu then selecting the "System Configuration Menu". This option displays the individual modules in the racks and the bus addresses in the genius configurations. For more information on setting up the PLC configuration see the chapter devoted to Configuring your system.

From the System Configuration display, move the cursor on top of the module to be viewed and press <Enter> to view a menu. Use the menu to view the I/O that is assigned to that particular module. The English names, if defined are shown or may be added for each circuit on this module.

# On-Line Program Changes

The State Logic program can be changed without halting the State Engine. Changes to the State Logic program are made one Task at a time. To make State Logic program changes while the program is running, select the **On-line Modify** option from the main menu and then select the Task to edit from the list of Task names displayed.

## Restrictions

There are some restrictions to how much modification can be done with the On-Line Modify option. The user cannot add or change digital I/O or Internal Flags. Variables cannot be changed to save over halt. The Project Management features cannot be used (copy, import, etc.). The user cannot change any of the system configuration data, i.e. auto run. New states can be added to the program. The user can add up to five states to any Task

### Memory Usage

Every time the On-Line Modify feature is used, an additional portion of memory is used. Repeated use of the On_Line modify feature may cause an error message indicating that there is not enough memory available. Using the "Translate and Download" feature from Program Mode will free up all the memory, and allow additional use of On-Line modify.

# Hints for Using ECLiPS Features

## How to Use the ECLiPS Menus

There is one starting menu for each mode of ECLiPS. Press <F3> to view the main menu when in Program Mode or Debug Mode.

Options are selected from all ECLiPS menus in the same manner. Use the up and down arrow keys to highlight the option and press the <Enter> key. A quicker method of making a selection is to press the highlighted letter of the desired option. Each option has a letter (usually the first letter) that is displayed in a different color or highlighted (for monochrome monitors).

## Using ECLiPS Hot Keys

The most popular menu options may be selected without using any menus by pressing a Hot Key. Any Hot Keys are listed to the right of the option on the menu where the option is displayed. A full listing of all key functions is available at any time by pressing the help key <F1> twice.

Appendix A has a table of Hot Keys. Notice that the Hot Keys are assigned in a manner to make it easy to remember their functions. All of the Hot Keys which refer to options from the Project Menu use the <F2> key. The <Ctrl> key plus letters refer to options from the List Menu or for Debug Mode selections, for example <Ctrl + D> lists the Digital Points defined in the program. The <F4> key is used for Define Menu options; the <F5> key is used for the Find Menu options, <F6> for Add Menu options, and <F8> for the Text options.

## ECLiPS Word Processing Functions

A program developed using ECLiPS is simply a document of ASCII text. The ECLiPS editor is fundamentally a word processor with specialized functions for editing a State Logic program.

The Text option offered on the Program Mode Menu is very useful for manipulating blocks of program text. The Text functions are used to copy, move, and delete blocks. These functions are very useful for manipulating blocks of the program.

When any of the Text functions are used, the block defined is saved in memory. This memory location is called the Paste Buffer. The Paste Buffer always contains the last block highlighted for any of the Text operations. The contents of the Paste Buffer are 'pasted' into the program at the cursor location by pressing the <Ctrl + U> keys. The PASTE function may be used to move or copy blocks of text. The PASTE function is also useful for copying blocks from one program to another.

Use the following steps to copy a block from one program to another program.

Press <F8>, the Hot Key for the Text functions
Highlight the desired block by using the cursor movement keys
Press the <Enter> key and choose the COPY option from the menu
Press the <Esc> key to cancel the operation, the paste buffer now has a copy of the block
Load another program into ECLiPS
Press <Ctrl + U> to enter a copy of the block at the cursor location

## How to Use ECLiPS Lists

ECLiPS uses lists to display the elements of the control program including I/O points, keywords, filler words, and variable names. The arrow keys and the <Page Up> and <Page Down> keys are used to move through the list. Also use <Ctrl + End> and <Ctrl + Home> to move to the end or the start of the list.

The elements of the list are displayed in alphabetical order. One method to quickly find an element is to enter the letters of the element. When a letter key is pressed, the first element starting with that letter is highlighted. Pressing another letter key causes the first element starting with those two letters to be highlighted. It is possible to enter all of the letters of the element to highlight it, but usually a few letters brings the highlight close to the target element.

Generally the <Ins> key is used to add, the <Del> key to delete, and the <Esc> key to exit the list. Lists displayed using the "List" option from the Program Mode main menu, also allow the highlighted element to be entered directly into the program at the current cursor location when the <Enter> key is pressed.

# Chapter 6 Program Instructions

This chapter describes how to structure the State Logic program instructions. The hierarchy structure of the program is described, and there is a quick reference list of all of the types of both Functional and Conditional Terms. These lists are followed by more complex examples and detailed explanations of Conditional and Functional Terms. Also included are sections on mathematical calculations, variables and data types. Finally is a list of grammatical rules and an explanation of filler words are included.

# Program Structure

There is a hierarchy in the structure of the State Logic program. Each program is divided into one or more Task Groups, each Task Groups may have one or more Tasks, each Task is divided into one or more States, etc. The figure below lists each element of the hierarchy in descending order of significance.

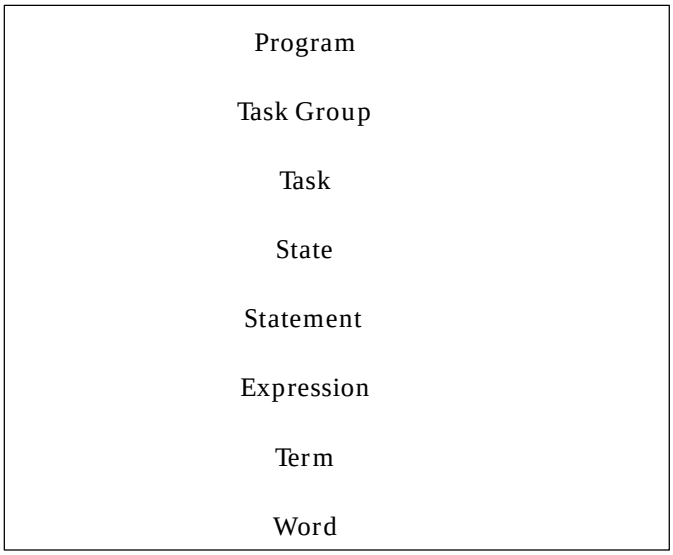


**Figure 6-1. ECLiPS Program Hierarchy**

This hierarchy of Tasks, States, and Statements are explained in sections 2 and 3 of this manual. A program is a collection of Tasks Groups, a Task Group is merely a collection of Tasks and is used only as an organizational convenience. Tasks are a collection of States which describe a sequence of actions.

There may be many Tasks all executing simultaneously. Each State is described by one or more Statements and each Statement consists of Expressions. Expressions are constructed from Terms which are composed of words.

## Task Groups

Task groups are simply a collection of Tasks. Each Task Group can have up to 60K bytes of the State Logic program. There may be up to sixteen Task Groups in a program. The program instructions for each Task Group is stored in a separate file that has the project name and the extension TG followed by 1–F. The first Task Group for the project PRESS is stored in the file PRESS.TG1.

## Tasks

In the program each Task begins with the keyword, **Task**, followed by a colon then a Task Name.

**Task: Assembly**

Each Task includes all of the States from the start of the Task to the beginning of the next Task.

The Task name may be followed by the keyword, **Start_In_Last_State**, which indicates that this Task is to restart with the State that was active when the program stopped running. This feature works the same whether the program was stopped deliberately or by a power failure.

**Task: VatCooling Start_In_Last_State**

## State

Similarly, each State begins with the keyword, **State**, followed by a colon then a State Name.

**State: Attach_Arm**

Each State includes all of the Statements from the start of the State to the beginning the next State.

After the State name the **Max_Time** keyword followed by a number may be used. This keyword causes a diagnostic message to be displayed in either the ECLiPS or OnTOP on–line Terminal Log screens, if the State is active longer than the number of seconds indicated by the number following the **Max_Time** keyword.

## Statements

Statements, like English sentences, are terminated by a period. The first Statement of a State begins right after the State name and includes all of the expressions appearing before the period.

During program execution each Statement of the active State is executed in order starting at the first Statement in the State. The only exception occurs when a **GO** term is executed. Nothing else in a State is executed after a **GO** is executed.

**If the ForwardLimitSwitch is on, go to the Idling State.**

## Expressions

There are two types of **expressions**, Conditional and Functional.

Functional expressions describe some action that the controller executes. A Functional Expression may be the only expression in a Statement or may be accompanied by a Conditional Expression.

**Turn on HeaterCoil_1.**
If TankTemperature is greater than 45.6, **go to the Cooling State**.

**Figure 6-2. Functional Expressions in Bold Type**

Conditional Expressions describe a requirement which must be satisfied for the functional expression in the same statement to be executed. A Conditional Expression must always be used together with a Functional Expression.

**If TankTemperature is greater than 45.6,** go to the Cooling State.
Go to the Collection State, **when 5.5 seconds have passed**.

**Figure 6-3. Conditional Expressions in Bold Type**

## Terms

Functional Expressions are comprised of one or more Functional Terms and Conditional Expressions are made of one or more Conditional Terms. Logical AND or OR words are used to join conditional terms.

**If 35 seconds have passed and Parts_Count > 5** then go to Restart.
**If ForwardProx is on OR ForwardMotor is on AND Torque > 95,** go to ShutDown.
Go to ClosePress, **if 3.5 seconds passed and (autoPB is on or startPB is ON)**.

**Figure 6-4. Examples of Multiple Conditional Term Expressions**

**Turn on DrainValve, DrainPump, and FlowLight.**
If InComingSwitch is on, **add 1 to PartsCount and go to Clamp**.
Start_Pid TankTemperature, write “Tank Temp Auto”, and go to Testing, if AutoPB is on.

**Figure 6-5. Examples of Multiple Functional Term Expressions**

## Words

All words are classified into one of three categories: Keywords, Names, and Filler words. Words are separated by spaces. Multiple words can be simulated in a single word using the underscore character, as in **Forward_Solenoid,** or mixing upper and lower case letters, as in **ForwardSolenoid.**

ECLiPS comes with a set of keywords already defined. The programmer can define synonyms for the keywords and even change the default keywords. Because of this feature the control program can actually be written in another language.

All filler words are ignored when the program is translated into a control program. The keywords and names are the only words which affect the execution of the program. Filler words are used only to make the program more readable.

# Functional Terms

This section explains the various functional terms. The first display shows a quick reference list of the functional terms with an example of each term. The rest of the section describes the syntax and details of using each of the terms in the program.

**Table 6-1. Functional Term Quick Reference List**

| TYPE | EXAMPLES |
|---|---|
| Turn On Digital I/O | **Actuate** DropGateSolenoid. |
| Assign Variable and Analog Values | **Make** ExtrusionLength = 23.45.<br>**Make** VatHeater = 75.5.<br>**Make** WarningString = "Lubrication Reservoir LOW". |
| Perform Calculations | **Add** 1 to PartsCount.<br>**Make** SlabArea = SlabHeight * SlabWidth.<br>**Make** ResponseLevel = SIN(2*RadianAngle) + 75.3. |
| Transition to Another State | **Go** to EStop. |
| Change State of Another Task | **Put** the FlowControl Task into the NormalFlow State.<br>**Suspend_Task** the CutterControl Task.<br>**Resume**_Task the Cutter_Control Task. |
| Send Data Out the Serial Port | **Write** "Tank Full" to OperatorDisplay. |
| PID Control | **Start_PID** TorchTemperature.<br>Stop_PID WaterFlow. |
| Execute a Perform Function | **Perform** Time_Counter with<br>Action='A' Integer_Variable=NumberOfMinutes<br>TimeInterval='M'. |

## Turning ON Discretes (Actuate Term)

The Actuate Term is used to turn on Digital Points and Internal Flags. All of the digital points are OFF by default. Transition to a State that is not turning an output ON has the effect of turning the output OFF. There is no keyword that turns an output OFF.

DEFINITION: This Term starts with the keyword **Actuate** followed by one or more discrete names.

**Actuate** the Ready_Light.
**Start** Pump_1 and Pump2.
**Energize** Clamp_1, Clamp_2, Clamp_3 and Clamp_Flag.

## Assigning Analog and Variable Values

To assign values use the Make Term, Math–Assignment Terms, Set_Bit/Clear_Bit Terms. These Terms assign values to variables and analog I/O points.

## Make Term Definition:

The Make Term is used to assign a value to a variable or analog I/O point. The Term starts with the keyword **Make** and is followed by a variable or analog name, the keyword **equal**, then a number or a calculated value.

```
Make Flow_Setpoint equal 25.
Make Valve_Control = 67.89.
Make Total_Defects equal Temperature_Failures + Stress_Failures.

Make Output_String equal "Enter setpoint now".
Make Test_Character = '$'.

Make Tank_Level_PID Bias equal 34.456.
```

See the section "Calculated Values" for a description of how to do mathematical calculations.

## Math–Assignment Term

For simple mathematical calculations where a simple operation is performed to a numeric variable use the Math–Assignment Terms. These simple four function math terms are **ADD, SUBTRACT, MULTIPLY, and DIVIDE.**

The Add Term is the keyword **Add** followed by a number or variable name then a variable name.

```
Add 1 to Parts_Count
Add Second_Shift_Parts_Count to Total_Parts_Count
```

The Subtract Term is the keyword **Subtract** followed by a number or numeric variable name then a variable name.

```
Subtract 2.78 from Starting_Value
Subtract Tare_Weight from Test_Weight.
```

The Multiply Term is the keyword **Multiply** followed by a variable name then a number or variable name.

```
Multiply Parts_Lost by 2
Multiply Machine_Strokes by Strokes_Per_Cycle
```

The Divide Term is the keyword **Divide** followed by a variable name then a number or variable name.

```
Divide Right_Side_Length by 4.5
Divide Box_Volumn by Volumn_Adjustment
```

## Set_Bit/Clear_Bit Term

The Set_Bit / Clear_Bit Term is used to set an individual bit to either 1 or 0 in an integer variable. First the **Set_Bit or Clear_Bit** keyword is used then the variable name followed by the zero based bit number.

```
Set_Bit Transfer_Status 2
Clear_Bit Tac_Register 0
```

## Changing Active States Term

A Task can change its own State and can also change the current State of another Task. Changing the State of another Task is one way that Tasks can coordinate their activities.

The GO term is used by a Task to transition to another State. Only the Go and the State name are mandatory. All other words are optional. The Go may appear in any Statement but there may only be one Go per Statement.

```
Go to the Forward_Motion State.
Go EmergencyStop.
```

As soon as the GO term is executed, the new State becomes the active State for that Task. No more terms are executed in the old State and the new State is seen as the active State by other Tasks during the same program scan.

Tasks control other Tasks by merely setting the Task to a new State value.

```
Put the Assembly_Control Task into the Emergency_Stop State.
```

In this example, put is a synonym for make and into is a synonym for equal.

The Suspend_Task and Resume_Task Keywords are also used to change the current State of a Task. The Suspend_Task keyword saves the current State and puts the named Task into the Inactive State. The operating system gives every Task a State named Inactive. When a Task is in the Inactive State, this Task performs no activity and the only way to exit this State is for another Task to change its current State. The inactive State can also be used with the GO keyword.

The Resume_Task keyword causes the named Task to go to the State that was active before being suspended. If the task has not been suspended, then the Resume_Task keyword will cause the task to become inactive.

```
If Water_Level is above 45.5 feet then Suspend_Task Fill_Tank.
If Water_Level is below 43.8 feet Resume_Task Fill_Tank.
```

IMPORTANT: Make sure that both the terms changing the State of another Task execute for only one scan. These Terms, when placed in a Statement that is executed every scan, may cause some unexpected results.

## Sending Serial Data (Write Term)

The Term to send data out a serial port is the keyword Write followed by data to send inside double quotes. Optionally a communications port name may be specified following the data to be sent. If no port name is specified the data is sent to the programming port to be displayed by OnTOP or ECLiPS. See the chapter on the Serial Communications Module for more information about serial communications.

```
Write "Push Start Button" to Operator_Control.
```

The serial data can be a mixture of typed text, variable values, ASCII control characters, and formatting characters.

The typed text are any characters entered directly from the keyboard. The text may include carriage returns so that several lines can be entered in one Write Term. Multiple line messages can be formatted in the program exactly as they appear on an terminal screen.

```
Write "
Opening Operator MENU

1. Change Today's Date
2. Change the Current Time
3. Engage Startup Procedure
4. Restart the Process"
   to Operator_Panel.
```

The menu from the Write Term above appears on the operator screen just as it does in the program. The limit of the number of characters between the quotes is 512, which is about 7 full (80 character) lines of text.

**Variable values** are sent out the port by preceding a variable name with a "%". Any variable type, including analog I/O values but excluding discrete values, can be sent out the serial port.

```
Write "Current parts count is %Part_Count." to Operator_Terminal.
```

If the variable, Part_Count, has a value of 10 at the time the Write Term above is executed, the following line is displayed on the screen connected to the port named Operator_Terminal.

```
Current parts count is 10.
```

Formatting Characters that are used with the Write Term follow:

**%NOCRLF –** Write Terms always send a carriage return line feed pair following each message. Use this formatting feature to suppresses these terminating characters.
**%CRLF –** sends a carriage return line feed character pair
**%CRLF(X) –** X number of carriage return, line feeds
**%CLS –** Clear the Screen, sends 25 carriage return, line feeds
**%SPACE(X) –** X number of spaces

Control Characters are embedded in the string of characters. Embedded control characters have many uses, controlling screen displays for dumb terminals and PCs using ANSI.SYS device driver, some serial devices use control characters for special function control.

**%CHR(X)** – The ASCII character for the value in the "()" is sent
**%#X** – The ASCII character for the hexadecimal value X is sent

The " | " character is used to place two words together without any spaces in between them. For example, "%Pressurepsi" would look like one long variable name to ECLiPS and would yield an error message. But "%Pressure | psi" would yield the desired result of the value directly followed by the character string, "psi".

To send a double quote sign use "%#22" or "%CHR(34)". To send per cent sign use "%%". All other keyboard characters are sent by simply typing them between the quotes.

## PID Loops Control Terms (Start_PID, Stop_PID)

PID Loop control Statements start with the keywords Start_PID or Stop_PID, followed by the PID Loop Name. If stopping a PID loop, a value which sets the value of the control variable can follow the PID loop name.

**Start_PID** Oven_1.
**Stop_PID** TankLevel.
**Stop_PID** KILN5 with 456.29.

When the PID loop begins execution it uses parameters specified in the PID initial values form accessed through the DEFINE or LIST menus. See the chapter on PID loops for more information.

**IMPORTANT:** Make sure that the Start_PID Term is not executed repeatedly every scan by changing State after starting the loop operating. The PID loop will not execute as expected when it is constantly restarted. For more information on PID Loops see the PID loop chapter.

### Change Serial Port Configuration Term

The Term to change the configuration of a serial port is the **Set_Commport** keyword followed by a port name and then a list of parameters and their values. See the chapter on the Serial Communications Module for more information about serial communications.

This Functional Term is automatically entered into the program by using the "Communication Ports" option on the LIST menu. Select the port and press the right arrow key "–>" to change the configuration options. When configuration is complete, press <Enter> when the desired port is highlighted, then select the "Insert Reconfiguration Data for the Port" option from the next menu. The entire Term is entered into the program at the current cursor location.

These port parameters can also be changed by using the "System Configuration" option from the DEFINE menu. When setting the serial port parameters using the "System Configuration" option the parameters are changed program is downloaded and begins execution. The Set_Commport keyword method is used to change the settings from the program.

### Perform Function Term

This Term is the **Perform** keyword followed by the function name, the keyword with and then a list of parameters and values. This Term is entered into the program automatically by ECLiPS at the end of the State where the cursor is located. First select the "Add" option from the menu, then the "Add a Perform Function" option. Fill in the blanks that are displayed after selecting the function desired. See the chapter on perform functions for more information.

## Conditional Terms

Conditional Terms are used to test for conditions of discrete, numeric, analog, time and character values and also to test when input has been received in a serial port. Conditional Expressions must be true for the Functional Expression in the same Statement to be executed. Normally the program is constructed so that the Functional Expression that is dependent on a Conditional Expression is a transition to another State.

**Table 6-2. Conditional Term Quick Reference List**

| TYPE | EXAMPLES |
|---|---|
| **Digital** | If PressDownSwitch is ON . . . |
| **Time** | If 3.5 seconds . . . |
| **Relational** | If TankLevel is greater than 45.67 . . . |
| **Current State** | If the CleanInPlace Task is in the WaterRinse State . . . |
| **Serial Input** | Read OperatorInput from OperatorDisplay . . . |

## Digital Conditional Term

The Digital Conditional Term tests the status of discrete memory types (%I, %Q, %G, %T, %S) and the internal flags. To program this type of test use the IF keyword followed by the name of the discrete point then followed by the keyword ON or OFF.

**If** Forward_Limit_Switch is **on** . . .
**If** Part_Ready_Flag is **off** . . .

Several digitals can be specified in the same expression joined by AND or OR keywords as follows:

**If** Top_Limit_Switch or Bottom_Limit_Switch and Counter_Weight_Switch are **OFF** . . .

The ANDs are executed first when both ANDs and ORs are in the same expression.

## Timer Conditional Term

The Timer Conditional is a number or variable followed by the keyword SECONDS (must be plural). The timer has a resolution of 1 / 100 of a second and the value used to indicate the number of seconds can be a floating point number.

**If** 3.76 **seconds** have passed, then . . .

An integer variable can also be used to specify the number of seconds. The value of the variable indicates the number of hundredths of a second, so

that a variable value of 100 would indicate a time of 1 second.

**If** Wait_Time **seconds,** then . . .

Timers always refer to the amount of time that the State has been active. A common mistake is to assume that the timer starts when the term before it becomes true.

**If** Track_Monitor is **ON** and 5.3 **seconds** have passed . . .

The timer above refers to the time that the State that it is in has been active and is not influenced by the condition of the Track_Monitor.

The timer number must be in the range of 0.01 to 600.00 seconds which is a maximum of 10 minutes. When using an integer variable, the variable value must be a positive value.

There are several ways to make a timer that uses a period of time greater than 10 minutes. The common methods use State transitions to reset a State timer.

```
State:  Heater_On_One_Hour
    Actuate Vat_Heater.
    If Ten_Minute_Counter is >= 6, go to Start_Process State.
    Wait 600 seconds then go to CountMinutes State.

State: CountMinutes
    Add 1 to Ten_Minute_Counter and go to Heater_On_One_Hour State.
```

Also see the Time_Counter Perform function for a description of other ways to handle timing situations in State Logic programs.

## Relational Conditional Term

Relational Terms test variable and analog values. The Term is a value, followed by a relational operator, then another value. The values tested can be numbers, calculations, variable names, and analog names.

```
If Parts_Count = 500 . . .
If Flow_Meter_Input is above Flow_High_Limit . . .
If Canister_Pressure – Atmosphere  <=  Pressure_Limit – Safety_Margin . . .
If String_Entry equal "Formula 1" . . .
If Test_Char is not_equal to '@' . . .
```

See the section, Mathematical Calculations, for a discussion on how to use calculations with Conditional Terms.

## Current State Conditional Term

The Current State Conditional is a Task Name followed by the keywords EQUAL or Not_Equal and then a State Name. This conditional is used to test the current State of another Task.

```
If Pump_Monitor Task is in the Backwash State . . .
```

A Task can test the current State of any other Task. This term is one of the main ways that Tasks can coordinate their activities.

## Complex Conditionals

The Conditional Terms can be combined to form a Conditional Expression using the keywords AND and OR keywords. The AND Terms have lower precedence and are therefore executed first. The order of execution can be changed by use of parenthesis and parenthesis can be nested. The keyword NOT can also be used preceding the conditional term.

**If** Hydraulic_Pump_Control Task **is** in the Over_Pressure State or Hydraulic_Prssure is **above** 23.56 . . .

**If** 1 seconds and **not** Temperature_Setpoint **greater** than 4.67 / Settling_Value . . .

**If** Spin_Drive is **ON** and (Pour_Ladle is **not_in** Pouring State or **not** Mold_Number is **above** 67) . . .

## Character Input Conditional Term

The syntax for this conditional is the keyword READ followed by a variable name. This conditional is true when a character input message is completed. The character input is stored in the variable listed.

Optionally this conditional can specify the port from which the input is received. If this option is used the keyword FROM follows the variable name and then a communications port name is listed.

**Read** Menu_Choice **from** Operator_Station, then **go** . . .

The above example reads data into the variable named, Menu_Choice, from the communications port named Operator_Station.

A GO Functional Term must always follow the character input conditional and there cannot be any other Terms in the Statement besides the READ term and the GO term.

If two READ terms for the same port are both currently active (both in an active State) it is unknown which conditional receives the message from the port. The program should be written such that all READs for a port occur in the same Task, assuring that two READs for the same port are not both active at the same time.

The types of variables used with the Read are:

Integer Variables

Floating Point Variables

String Variables

Character Variables

If the type of data received does not match the variable type, the input is ignored and the conditional is not satisfied. An example of invalid data is entering string of characters to a numeric variable.

The input is completed and the Conditional Term is true, when an end of message character is received at the port. The default end of message character is the carriage return, so that normally the input is completed when the <Enter> key is pressed. The end of message character may be changed using the serial port setup forms. See the Serial Communications Module chapter for more information on serial communications.

Input to a character variable is complete as soon as one character is received, so that a character is stored and the GO executed as soon as any character is received.

IMPORTANT: Character variables cannot be used to receive input through the programming port when connected to ECLiPS or OnTOP.

# Mathematical Calculations

Mathematical calculations are used in Functional Terms as in this assignment Term:

**Make** Pointer_Position = Last_Position * (Forward_Pressure + 345.8)

and in Conditional Terms as in this comparison term:

**If** Advanced_Magnitude < SIN(Current_Angle) / 45.6 go to Reposition State.

Numerical expressions may be much more complicated using any of the operators in any order and nested in parenthesis to change order of evaluation or make the expression more readable. Up to 18 levels of parenthesis may be used.

## Operator Precedence

Operators are executed according to their precedence. The operators with the lowest precedence number are executed first. Operators with the same precedence are executed left to right. Use parenthesis to change the order of execution. See the operator keyword table for the precedence of each operator.

# Variables

Variables are used to store some information in memory. All variables are identified by a unique name that is assigned when the undefined words in the program are defined. Each variable can be configured to save the value over a power cycle or be initialized when the program is started. There are two main categories of variables, ASCII and numeric. Calculations may refer to the value stored in any of the numeric variables.

## ASCII Variables

The ASCII variable types are Character, and String. Character Variables store one character and use one byte of memory. String Variables store up to 80 characters and use 82 bytes of memory, since every string is terminated by a null character, (00), and memory is used in two byte words.

String Variables store any ASCII characters. Control characters are used in the string variable by using the % followed by the # and then two digits that are the hexadecimal number for the ASCII character, for example:

```
Make Test_String equal “abc%#1Bxyz”.
```

This example is an assignment to a String Variable to store the characters abc the escape character and then xyz.

## Numeric Variables

The numeric variable types are Integer, Floating Point, Time, and Analog. The numeric data types are described later in this section.

### Analog Variables

Analog Variables are the values of the analog I/O connected to the system. Analog values are floating point data type if the channel is scaled and integer if unscaled.

### Time variables

Time Variables store the current time information from the system clock. The reserved time variables are **second, minute, hour, day, day_of_week, and month**. These variables are integer values.

These variables are read only variables meaning that they cannot be changed from the program or from the debug mode CHANGE option. To change the system clock, use Logicmaster 90 configuration package to change the CPU clock values. Access to the clock values are from the Logicmaster CPU configuration option.

**IMPORTANT:** Changing the clock while the program is running may cause some timing functions in progress to work erratically.

### Integer Variables

Integer variables store 2 byte integer values.

### Floating Point Variables

Floating Point variables store 4 byte IEEE format floating point values.

### Numeric Data Types

There are two numerical data types integer and floating point. Integer data is limited to the range of –32768 to 32767. Floating point data is limited to the range of +/–1.2E–38 to +/–3.4E+38 with an effective precision of seven decimal digits.

Integer constants are whole numbers in the range –32768 to 32767. Integer constants can be specified in hexadecimal format by preceding the number with '#', i.e., #77FF. Floating point constants are numbers using decimal points, numbers outside the range for integers, or numbers using scientific notation.

The numerical data types may be mixed freely within expressions. If any of the operations in an expression require floating point notation, all of the data elements are converted to floating point values. If a floating point value is assigned to a variable of integer variable type, the floating point value is converted to an integer value observing the following rules:

1. All values are rounded to the nearest number.
2. Values outside the integer range are clipped to –32768 or + 32767.

Floating point operations require more time to perform than integer operations. Therefore, refrain from floating point operations as much as possible if response time is critical to your application.

## Grammatical Rules

- Every Task must begin with the word "Task:" followed by the Task name.
- Every State must begin with the word "State:" followed by the State name.
- Every Statement must end with a period.
- Every Statement must have a functional expression.
- Only one "Go" is allowed per Statement.
- Only one "Read" is allowed per State.
- If a "Read" Term is used in a Statement, it must be accompanied by a "Go" in the same Statement. There may be no other Terms in the Statement.

## Filler Words

Filler words have no functionality, i.e. they do not change the meaning of any of the statements in which they appear. Filler words are only be used to increase the clarity of the English text. For example, "go to the Motion State" looks better and sounds better than "go Motion". To some programmers, however, typing fewer words is better.

# Chapter 7 Perform Funtions

The Perform functions implement operations which are more complicated than the common State Language Terms. To use a Perform Function, fill in one of the forms that ECLiPS provides. Once the form is completed press <F9> to save the data and ECLiPS then enters the code for the Perform Function at the current cursor location. To access the forms select the “Perform Functions” option from the ADD menu.

The State where a Perform function appears should usually be structured so that the Perform is executed for only one scan. A common error in using Performs, is to structure the program to execute a Perform function many times i.e. each scan.

ECLiPS presents a form to enter the parameters for the Perform functions. The forms to enter the functions provide information about each of the parameters. The specialized performed function at the end of this chapter use slightly different forms. The column headings of the forms are as follows:

**Parameter Name** **Type** **Use** **Required** **Actual**

The meaning of each of these columns is described below:

| | |
|---|---|
| **Parameter Name** | identifies the parameter described in this row of the table. |
| **Type** | specifies the parameter data TYPE and may be integer(I), floating point(F), character(C), string(S) or digital(D). |
| **Use** | Specifies whether a variable, constant, or both may be used for this parameter |
| **Required** | Specifies whether this parameter is required in the function call. The parameters that are not required are at the end of the parameter list. No parameters may be entered following one that has not been used. |
| **Actual** | This is where the variable or constant for the parameter is entered for this particular function call. |

# Table Functions

The table functions are designed to use data in a table, or array type fashion. A Table is a two dimensional array of values made up of rows and columns. It should be noted for all the Table functions the row number comes first followed by the column number.

There can be a maximum number of 100 Tables, each assigned a unique number from 1 to 100. These tables can be of any size until the maximum amount of memory allocated for Table use is consumed. There are 20K bytes of memory reserved for use with table functions. The size of a table is determined by the number of rows and columns it is defined to have.

There are four TYPEs of data that ECLiPS uses in tables, they are: floating point numbers (float or F), integer numbers (integer or I), sequence of characters (string or S) or binary numbers (digital I/O status or D). The table and all of the data in the table must be defined to be of the same TYPE. The Define_Table function is described in detail later.

There are four Swap_Table_Value functions, one for each of the four TYPEs of data. They are named Swap_Table_Value_Int, Swap_Table_Value_Float, Swap_Table_Value_Dig, and Swap_Table_Str. The swap functions will either: a) write from a variable into a Table element, or b) read from a Table element into a variable. The variable and the table must be of the same TYPE of data. The Swap_Table_Value functions are covered in detail later.

The Init_Table functions is designed to initialize a table by writing a number of values to the table in one perform function. Up to 28 data elements can be stored in a table with one perform function. There are three data TYPE specific Initialize functions, they are: Init_Table_Integer, Init_Table_Float, and Init_Table_Digital. There is no initialization function for a string table. Detail descriptions of the Init_Table function follow later in this section.

The Copy_Table_to_Table function is used to copy data from one table into another table of the same TYPE. The Copy_Table_to_Table function is described in detail later in this section.

ECLiPS and the State Engine perform error checks on all table functions to insure that all actions performed are consistant and legal. They will to check that the data TYPE is consistant, that the row number and column number being used are within the defined limits of the table, etc. The errors will be detected during download or when the program is run. The most common place for the errors to show up is at run time.

## Define_Table

Every table must be defined before it can be used. Defining the table will: a) assign a number to the table (1 to 100), b) determine the size of the table, and c) determine the type of data in the table. This function will provide a form asking for the following parameters:

| | |
|---|---|
| Number_of_table | the Table number from 1 to 100 |
| Type_of_table | the type of variables (I for integer, D for digital or F for float, S for String) stored in the Table elements |
| Number_of_rows | the number of rows of the table |
| Number_of_columns | the number of columns |
| Save_value_over_halt | indicates whether the Table should be saved through a halt-run cycle. Enter Y or N. |

At State Engine run time, a non-critical error with a message will be generated if the Table number specified has already been defined. The Table will not be re-defined and the original definition will be retained. This problem may occur if the define table function is in a state that executes (or scans) more than one time. The only other non-critical run time error that can occur is if the table number specified is greater than 100.

## Entering and Retrieving Table Values

There are four Swap_Table_Value functions that are exactly the same except they work on the four different types of Tables.

Swap_Table_Value_Int

Swap_Table_Value_Flt

Swap_Table_Value_Dig

Swap_Table_Value_Str

The Swap functions are used to exchange data between a table and a single variable. Values can be transfered by either a) write a value from a variable into a Table or b) read a value from a Table element into a variable. There are four distinct functions one for each different data TYPE. The TYPE of data in the table and the variable must match.

When the User selects one of these functions from the Perform menu the following information will be requested:

| | |
|---|---|
| Number_of_table | the Table number from 1 to 100 |
| Type_of_operation | (R or W) R for read from a table into a variable or W to write from a variable into a table. |
| Row_number | the row number of the element to be read from or written into |
| Column_number | the column number of the element to be read from or written into |
| Variable | the name of the variable that will exchange data with table |

The State Engine will generate run time non critical errors if the TYPE of the Table does not match the TYPE of Swap being used, or if the row and column numbers are out of range for the selected Table.

## Initializing Tables

The three Init_Table functions will initialize a table with up to 28 values in one Perform function. There are three Init_functions one for each data TYPE. There is no initialization function for String Tables. The type of data that is being place in the table and the data type of the table must match. The three Init_Table functions are:

Init_Table_Int

Init_Table_Flt

Init_Table_Dig

Selecting one of these functions from the Perform menu will generate a form with the following parameters to be filled in:

| | |
|---|---|
| Number_of_table | the Table number from 1 to 100 |
| Row_number | the row number of the first element where the values listed are to be stored |
| Column_number | the column number of the first element where the values listed are to be stored |
| Number_of_values | the number of values that will be stored in the following consecutive Table elements |
| Value_1 | a constant (of the same TYPE as the Table) to be stored in the first Table element identified by the Row_number and Column_number |
| Value_2 | a constant (of the same TYPE as the Table) to be stored in the first Table element after the Table element identified by the Row_number and Column_number |
| | Each additional value must be consecutive. ie. you can not have Value_3, without Value_2 and Value_1. |
| Value_28 | a constant (of the same TYPE as the Table) to be stored in the last Table element identified by the Row_number and Column_number |

There can be up to 28 values initialized with each individual function and they can begin at any Table element location. The fill order is: first to fill across the row (left to right), when the row is full, the next row down will be filled.

The State Engine will generate run time critical errors if the Table selected does not match the TYPE of Swap being used, if the row and column numbers are out of range for the selected Table, or if the number of values added to the starting element position would go beyond the last element defined for the Table.

### Copy_Table_To_Table

The Copy_Table_To_Table function places one Table's values into another Table. The Tables must be of the same data TYPE and the Table to be *copied from* must be equal to or smaller than the Table to *copy into*. If the table to be copied from is smaller than the the table to be copied into, any data elements that are not copied into will be left unchanged.

Choosing Copy_Table_to_Table from the "Add a Perform" menu will provide a form asking to fill out the following parameters:

| | |
|---|---|
| Table_To_Copy_From | the number of the Table from which the values will be copied. Its row number and column number must be less than the other Table identified |
| Table_To_Copy_Into | the number of the Table the values will be written into. |

The State Engine will generate run time critical errors if the two Tables selected are not the same TYPE or if the Table to copy from is larger than the other Table.

### Table Uses

There are many uses for the Table functions. As an example, the Table functions are valuable in applications where the set up of parameters varies depending on the product under manufacture on the process line. Batch process recipes or flexible manufacturing assembly lines are examples.

The ECLiPS program can be written using English name variables for parameters throughout with statements such as:

If Oven_temp_1 is greater than Melting_point ...

used throughout the program description of the process. Then in a State, lets call it the Select_Product State, by using the Swap_Table_Value_Flt function, the variable Melting_Point can be made equal to one of the elements of Table 1, where Table 1 contains the parameters for this particular product run.

Using the Init_Table_Float, Tables containing parameters for each style of product that can be made on the line can be initialized. When the Operator selects a style of product in the Select_Product_Style State, the Copy_Table_To_Table function can be used to move those parameters into Table 1, the Table in which Melting_Point finds its values for this style and product run.

## BCD I/O Representation

### General

At times input and output devices are used for data entry or display that use BCD representation. Thumb wheel switches and LCD displays are possible examples.

The devices are connected either to digital inputs or digital outputs where 4 hardware inputs or outputs represent 1 digit of the display. The display or switch then uses a binary code from the four I/O to represent from 0 to 9. There are 16 total possible bit combinations (4 outputs or inputs represent 2 to the 4th or 16 possible combinations). The remaining 6 bit combinations are used for: minus sign, decimal point, and null or space character.

The two functions, BCD_In_Conversion and Output_BCD_Conversion allow the User to designate a series of consecutive digital inputs or outputs to be treated as if they are groups of 4 BCD digits. The function then translates between the I/O and a numeric variable, either integer or floating point.

## BCD_In_Convert

The BCD_In_Convert translates between a series of digital inputs and an integer or floating point variable. The inputs will be taken in hardware consecutive order, and the number of digits can be on more than one block or card as long as the cards have consecutive addresses. Each of the digital input lines must be defined with an english name.

**example:** a device that will provide a two digit number will have 8 digital lines or inputs, 4 for each digit. Each of those eight inputs must be defined with an English name.

Another factor for consideration is the order of significance of the digital inputs. Eclips treats **the first input as the Least Significant Bit of the Most Significant Digit**. For example, lets imagine the above device was going to transmit the number 34 to the controler. The eight inputs would transmit a 3 (0011) and a 4 (0100). Eclips treats the first input as the least significant bit of the most significant digit. In this case a 1. The next bit would be 1, then 0, and 0. These four bits make up the most significant digit. The next bit would be the first bit of the 4 or 0, then 0, then 1 and then 0.

**The inputs used to receive the digital number 34 if %I12 is the first input**

| | | | |
|---|---|---|---|
| %I12 | 1 | %I16 | 0 |
| %I13 | 1 | %I17 | 0 |
| %I14 | 0 | %I18 | 1 |
| %I15 | 0 | %I19 | 0 |
| | 3 | | 4 |

BCD uses the standard binary representation for the numerical digits 0 to 9. There is no true standard for the minus sign or decimal point character. Therefore the function has the provision for the User to optionally specify the hexadecimal number, #A, #B, #C, #D, #E, or #F, (where # means hexadecimal number to ECLiPS), that is the pattern for these two characters.

The function parameters are:

| | |
|---|---|
| Starting_input | the English name of the first digital input in the string of consecutive inputs that form the BCD digits. This input will be *the least significant bit of the most significant digit* |
| Number_of_BCD_digits | the number of BCD digits. The number of digital inputs in the string will be 4 times the number of digits. |
| Variable_name | the name of the variable to store the translated value |
| Variable_type | the data TYPE of the variable either integer (I) or float (F) |
| Minus_sign_pattern | a hexadecimal number from A to F that gives the pattern for a digit that should be recognized as the minus sign. |
| Decimal_point_pat | a hexadecimal number from A to F that gives the pattern for a digit that should be recognized as the decimal point. Note the minus sign and decimal point are optional. A minus sign can be specified without a decimal point, but if a decimal point is specified the minus sign must also be specified. |

The state engine will generate a run time error if any group of four bits does not equal either a number, from 0 to 9 or the minus sign or decimal sign pattern.

## Output_BCD_Convert

The Output_BCD_Convert translates between an integer or float variable and a series of digital outputs. The User will enter the name of the first digital output into the form as well as the number of digits that are stored in the variable. The User also specifies the TYPE and name of the variable that the value to be converted is stored in.

The outputs are in hardware consecutive order, and the number of digits can be on more than one block or card as long as the cards have consecutive addresses. Each of the outputs must be defined with an English name. Two output a two digit number eight bits would be required. Each of the eight bits must be defined.

The first digital output will hold the information for the least significant bit of the most significant digit. If the variable is holding the value of 56 two digit will be sent out with 8 bits of information, 5 (0101) and 7 (0110). The first output will hold the least significant bit of the 5 or 1, the next bits will be 0, 1, 0. The fifth bit sent out will be the first bit of the value for 6 or 1, the next bits will be 1, 1, 0.

**The outputs used to to send 56 with %q11 being the first output bit**

| | | | |
|---|---|---|---|
| %Q11 | 1 | %Q15 | 0 |
| %Q12 | 0 | %Q16 | 1 |
| %Q13 | 1 | %Q17 | 1 |
| %Q14 | 0 | %Q18 | 0 |
| | 5 | | 6 |

BCD uses the standard binary representation for the numerical digits 0 to 9. There is no true standard for the minus sign or decimal point or null (space) character. Therefore the function has the provision for the User to optionally specify the hexadecimal number, #A, #B, #C, #D, #E, or #F, (where # means hexadecimal number to ECLiPS), that is the pattern for these three characters.

The function parameters are:

| | |
|---|---|
| Starting_output | the English name of the first digital output in the string of consecutive outputs that form the BCD digits - all of the other outputs used must be defined. This will be the *Least significant bit of the Most significant digit*. |
| Number_of_BCD_digits | the number of BCD digits. The number of digital outputs in the string will be 4 times the number of digits. |
| Variable_name | the name of the variable that is to be translated and output |
| Variable_type | the TYPE of variable either I for integer or F for float |
| Minus_sign_pattern | a hexadecimal number from A to F that gives the pattern for a digit that should be recognized as the minus sign. |
| Null_character_pat | a hexadecimal number from A to F that gives the pattern for a digit that should be recognized as the null or space character. |
| Decimal_point_pat | a hexadecimal number from A to F that gives the pattern for a digit that should be recognized as the decimal point. |
| Number_decimal_dig | the number of decimal digits; the digits to the right of the decimal point, that should be output. Note: If the Number_decimal_dig parameter is used then the Decimal_point_pat is not optional and must also be used. |

Note: if the value is too large to display in the Number_of_BCD_digits specified, but there is enough room for all the digits to the left of the decimal point, those digits will be displayed and no error will be generated. If there is not enough digits for all the numbers to the left of the decimal point, the display will not be output and a non-critical error will be generated.

## Shift_Register

The Shift_Register function allows the User to shift values from one integer variable to another by a User selected number of bits.

The User can define up to 28 integer variables and connect them together to form a shift register. The maximum size for the shift register would be 448 bits (28 variables of 16 bits each). The contents of the shift register can be shifted by a selected number of bits (up to 64) to the right or left.

The shift can behave in a circular fashion or in a linear fashion. In a circular shift to the right the bits from the variable farthest to the right that are shifted off the shift register are placed into the first bit of the variable furthest to the left. If the shift is a fill type shift the bits fall off the end and do not circle back to the first integer. In the case of a fill type shift, the value (0 or 1) that is placed in the locations left empty by the shift can be specified.

The User defined function parameters are:

| | |
|---|---|
| Number_of_bits | the number of bits to shift the registers, 0 to 63 |
| Shift_direction | either R right or L for left. The direction of the shift. |
| Type_of_shift | either C for circular or F for fill, The way the shift will act when the last variable is reached. |
| Fill_value | either 1 or 0. The value that will be placed in the bits of the variables that have been shifter in a F or fill type shift. No meaning in a circular shift. |
| Variable_1 | the name of the first integer variable and therefore the variable farthest to the left in the shift register. |
| Variable_2 | the name of the second integer variable in the shift register. |
| Variable_28 | the name of the 28th or last integer variable and therefore the variable farthest to the right in the shift register. |

**Note:** only Variable_1 must exist, all others are optional.

# String Manipulation

## General

The String_Manipulation function is used to perform various functions upon a string variable. String variables can be up to 80 characters long and often are used for inputting data from an ASCII oriented device such as a bar code reader, or outputting to a similar device such as a scale or robot.

In many cases the 80 characters is not one piece of data but a series of sub-strings each containing unique data. Thus the ability to manipulate the large strings into smaller units and to combine small strings and numeric data into large strings is useful. ECLiPS provides a number of different functions to manipulate data into and out of strings.

ECLiPS will allow exchanging data between variables and the string. Numeric data will be converted to ASCII or into a numeric data type, depending on the action performed. String data types can be extracted and inserted as needed. In addition, ECLiPS facilitates searching for specific characters (match function) and testing the length of data strings (length function).

ECLiPS provides a form with the following parameters:

| | |
|---|---|
| String_name | The name of the string to manipulate |
| Start_character_num | The number of the starting character used in the string. The first character is 1. |
| End_character_num | The number of the ending character used in the string. The last character is 80. |
| Operation | A character code that defines the operation to be performed (see following section for definition) |
| Reference_variable | The name of the reference variable to be used in this manipulation. The variable type required depends on the Operation and the State Engine generates a critical error if a mismatch occurs. |
| Search_Character | The name of a Character_Variable or the actual character to be matched by this operation. This is an optional parameter which only needs to be entered when the match (M) operation is chosen. More information on Match follows. |

## Operations

ECLiPS uses the same form for all of the different string functions, the operations parameter specifies which operation will be performed at this time. The operation character is case dependent. In general, using upper case will write data from the string to a variable. Lower case, is used to insert a variable value into a string. The operation will also convert string values into numeric values when writing to the variable, and convert the numeric variable to ASCII code when writing to the string.

### ( E ) for Extract substring to string variable

If the Operation character is an E, the function will extract from the String_Name variable and place the data into the Reference_Variable as ASCII data. The Reference_Variable must be defined as String data TYPE. Starting and ending characters parameters determine what will be copied to the Reference_variable.

### ( s ) For store a string variable into the String_name variable

If the Operation character is an 's', the function will use the string variable named in the Reference_variable as a sub-string, and store those ASCII characters in the String_Name variable. The position of the sub-string is defined by the starting and ending character numbers. The characters in the string named as the Reference_value will be stored until the end of that string is reached or until the last character number in the main string is reached. If the sub-string (Reference_Variable) is too long the sub-string will be truncated.

### ( I ) for Extract from string and convert to Integer Variable

If the Operation character is an I, the function will extract the sub-string defined by the starting and ending character numbers and convert those ASCII characters into an integer value and store that value in the Reference_variable. Reference_Variable must be defined as an integer data TYPE.

### ( i ) For Convert integer variable to ASCII And Store In String

If the Operation character is an i, the function will convert the value stored at the integer variable named in the Reference_variable to ASCII and store that value into the String_Name variable at the location defined by the starting and ending character numbers.

### ( F ) for extract from string and convert to Float Variable

If the Operation character is a F, the function will extract the sub-string defined by the starting and ending character numbers and convert those ASCII characters into a float value and store that value at the float variable named in the Reference_variable.

### ( f ) for Convert float variable to ASCII And Store In String

If the Operation character is a f, the function will convert the value stored at the float variable named in the Reference_variable to ASCII and store that value into the String_Name variable at the location defined by the starting and ending character numbers.

### ( C ) for Concatenate or Add to String

If the Operation character is a C, the function will concatenate or add the string of characters named in Reference_variable to the main string. The resulting main string can not exceed 80 characters, so the addition of the Reference_variable characters will truncate any additions to keep the String_name variable to 80 characters.

### L for string Length

If the Operation character is an L, the function will calculate the number of characters in the string, and put that number into the integer variable named by Reference_variable.

### M for Match the Given Character with a Character in the String

If the operation character is an M, the function matches the character in the Search_Character parameter to the first character in the sub-string designated by starting character and ending character values. The position of the first match is returned in the Reference_Variable.

### Errors in General

The functions check to make sure when conversions to or from ASCII are performed that legal values will result and produce errors if they do not.

## Time Counter

This time function is designed to keep track of the elapsed time of an event. The time is stored in named integer variables that can be examined the same way that any other integer variable is examined. There are four type of time counters, each one being incremented at a different time period (tenth of a second, second, minute, and hour). There may be up to 30 each of tenth of a second, second, and minute counters and 10 hour counters active at any time during program execution.

The Time_Counter form that ECLiPS displays when this Perform Function is selected, has three parameters to specify the operation of the function:

| | |
|---|---|
| Action | A, E, H, or D to assign, enable, halt, or deallocate. see below |
| Integer_Name | The integer variable that is functioning as a timer |
| Time_Interval | T, S, M, H for Tenths, Seconds, Minutes, or Hours. see below |

Using a time counter requires a minimum of two perform functions. The first perform function is used to assign the integer variable named in the Integer_name parameter as a time counter. The Time_Interval that this time counter will use is specified at this time. After the integer variable is assigned the counter can be turned on by using the enable command.

The time interval does not need to be specified with the enable command, the counter will use the time interval specified in the assign perform function. Enabling a time counter will begin adding to whatever value is in the Integer_Name variable.

The value in the Integer_Name variable can be tested with conditionals, be assigned new values, or manipulated with mathematical operators at any time. As long as the counter is enabled value will be added to it at the specified rate.

**Action** Specifies how the function is applied to the counter. The data type of this parameter is character, and it may be specified by a variable or as a constant. A description of the possible choices for this parameter follow:

**'A'** - Assigns the specified variable to be a counter that is incremented as specified in the Time_Interval parameter. The variable is incremented only when it is enabled.

**'E'** - Enables a defined counter to begin counting the time interval specified when the counter is 'Assigned'.

**'H'** - Halts the counter from being incremented. The variable storing the time count maintains its value. To start counting again this counter must be 'Enabled'.

**'D'** - Deallocates the counter. The variable is no longer a counter, which frees up space for another counter of that time interval to be 'Assigned'.

**Integer_Name** Specifies the integer variable name to be used as a time counter. The named must be defined as an integer variable separately. The variable can be used as an integer in all ways, through the use of conditionals, assignments, and mathematical operators.

**Time_Interval** Specifies what time period must pass before incrementing the counter. The data type of this parameter is character and is specified by either a variable or a constant. This parameter is required only if the action parameter is an 'A'(Assign). For other actions this parameter may be left blank. A description of the possible choices for this parameter follows:

**'T' -** Specifies the counter to be incremented every tenth of a second.

**'S' -** Specifies the counter to be incremented every second.

**'M' -** Specifies the counter to be incremented every minute.

**'H' -** Specifies the counter to be incremented every hour.

**IMPORTANT:** The maximum value for each of these counters is 32767. After reaching the maximum, the counter continues counting, first becoming a negative value then returning to 0 and back positive again. If your counter goes over the maximum value, use the next higher counter type, for example use a minute counter instead of a second counter.

# VME Communication Functions

The VME functions are used to communicate with modules plugged into the VME rack. There are three VME functions, one for each of the data types that can be transmitted:

VME_Integer

VME_Floating_Point

VME_String.

Each function can either send or receive data depending on the ACTION parameter. The parameters for these functions are:

| | |
|---|---|
| Action | Determines whether the function reads or writes data, enter uppercase only, either R or W. |
| AM_Code | The address modifier which depends on the module type and slot location. |
| VME_Mem_Low_Word | Least significant memory address word for the data that is transmitted. |
| VME_Mem_High_Word | Most significant memory address word for the data that is transmitted. |
| Starting_Variable | The variable name that is the first memory location of data that is transmitted. |
| Total_Variables | The number of variables to be transmitted. |

The Total_Variables parameter is not used for the VME_String function. The VME_String function used to write a string uses the location designated by the Starting_Variable parameter and sends all of the characters stored in that string variable. When this function is used to read a string, 80 characters are read, starting at the indicated address location. The characters are placed into the string variable designated by the Starting_Variable parameter.

For information on addressing the VME module being accessed, contact the module manufacturer. You may also refer to the GE manual, "Guidelines for the Selection of Third-Party VME Modules", GFK-0448.

**Example**

| PARAMETER | TYPE | USE | REQUIRED | ACTUAL |
|---|---|---|---|---|
| Action | Char | Constant | True | *W* |
| AM_Code | Integer | Constant | True | *19* |
| VME_Mem_Low_Word | Integer | Both | True | *14336* |
| VME_Mem_High_Word | Integer | Both | True | |
| Starting_Variable | Integer | Both | True | *MotorSpeed* |
| Total_Variables | Integer | Both | True | *256* |

The fields that are filled in are *italicized.* The form filled in as above generates the following instructions entered into the program at the current cursor position.

```
Perform VME_Integer with
Action='W ',
AM_Code=19,
VME_Mem_Low_Word=14336,
VME_MeM_High_Word=0,
Starting_Variable=MotorSpeed,
Total_Variables=256.
```

# Specialized Perform Functions

All off the above functions have specific parameters which are passed to the function. These parameters are all chosen by filling in similar forms which specify parameter type and whether or not it is required. The following performs each have unique ways that the operations are specified. All of the Specialty Perform functions must be the only Statement in the State.

## Display Date and Time

This function displays the current date and time in the desired format. After choosing this option a form is displayed to enter the Name of the State that is created, the format of the display, the communications port to send the information and the State to branch to after the operation is completed.

## Get User Input

This function enters program text used to retrieve information through a communications port. A form is displayed for entering the following options:

| | |
|---|---|
| Current State | Name of the State that is created. |
| Clear Screen | Option of clearing the screen before the prompt is displayed. |
| Screen Message | Prompt telling operator to enter some information. |
| Input Variable | Variable that stores the input |
| Comm Port | Which port is used. |
| Branch To State | State that becomes active when this process is completed. |

## User Menu

This function will display a menu of up to 10 items and then wait for the user to enter a selection. If the selection is valid, it will branch to desired State for that selection. This is very useful when creating a user interface for the control program.

# Chapter 8 PID Loops

The Series 90-70 State Engine controller provides the capabilities of modulating control through the use of the PID algorithm. The Series 90-70 State Engine provides twenty PID algorithms that are continuously executed at user selected time intervals. These PID algorithms can be connected to field inputs and outputs, or interconnected in cascaded and other fashions to implement the desired control strategy.

This chapter describes how to set up the initial PID loop tuning constants and start the loop running. There are also sections describing how to tune the loop from both the Debug Mode tuning screen and by program statements. Finally there is a section describing the PID algorithm and its implementation in this product.

# Initializing and Starting PID Loops

There are several tuning constants and loop parameters associated with each of the twenty PID loops. This chapter explains how to initialize a PID loop, provides a definition of the parameters, and how to use program statements to start it running.

## PID Initialization Form

Each PID is initialized in the Program mode of ECLiPS during program editing. The starting parameter values are specified by filling in the PID form accessed through the LIST menu.

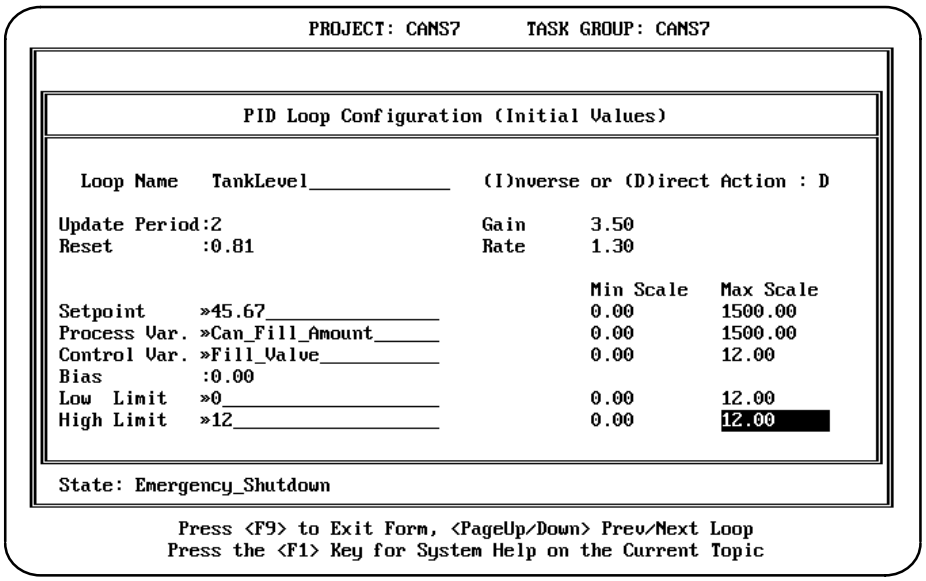
PROJECT: CANS7 TASK GROUP: CANS7
PID Loop Configuration (Initial Values)
Loop Name TankLevel_____ (I)nverse or (D)irect Action : D
Update Period:2 Gain 3.50
Reset :0.81 Rate 1.30
Min Scale Max Scale
Setpoint »45.67_____ 0.00 1500.00
Process Var. »Can_Fill_Amount_____ 0.00 1500.00
Control Var. »Fill_Valve_____ 0.00 12.00
Bias :0.00
Low Limit »0_____ 0.00 12.00
High Limit »12_____ 0.00 12.00
State: Emergency_Shutdown
Press <F9> to Exit Form, <PageUp/Down> Prev/Next Loop
Press the <F1> Key for System Help on the Current Topic

**Figure 8-1. PID Initialization Form**

Within the PID algorithm, all signals are treated as being 0 to 100%. Each input and the output and the high low limits, have scaling constants associated with them. Values given for this parameter are converted to a percentage of the range specified by the PID scaling constants.

If these constants are left blank at programming time, that input is assumed to be already 0 to 100%. Values given for this parameter are assumed to be a percentage which has already been scaled.

By using the scaling factors, the output can be scaled to have a live 0, that is go from -100% to +100%. This is a valuable tool at times when cascading PID's and the upstream PID needs the ability to overcome and move the downstream PID across its full range.

## PID Loop Paramters

The PID algorithm has several adjustable tuning constants. These include the Gain, Reset and Rate, the high limit and low limit, and a bias value. The bias is a value that is added to the output at all times and can be used to insure a minimum output from the PID.

The high and low limits set maximum and minimum values, within the scale maximum and minimum, that the controller output will not exceed. When these limits are reached, the appropriate status output is set, and anti-reset windup techniques automatically go into affect for that PID.

Every PID also can be selected to be either a direct acting or inverse acting controller. This is a parameter selected at programming time. A direct acting controller will integrate from 0 to 100% if the setpoint is greater than the process variable. A reverse acting controller will integrate from 0 to 100% when the process variable is greater than the setpoint. All other features are exactly the same whether the PID is in the direct or inverse mode.

The following is a summary of User selected values. These values are all entered in the PID Loop Configuration form displayed for initializing a PID loop. PID loops are initialized in program mode using the LIST options from the program mode menu.

**Table 8-1. PID Loop Parameters**

| Parameter | Keyword | Description |
|---|---|---|
| Action Direct or Inverse | Loop_Action | Set to D or I to make PID integrate from 0 toward 100% if the setpoint> process variable (direct acting) or process variable > setpoint (inverse acting.) |
| Update Time | Update | Time interval between updates for this PID. |
| Gain | Gain | Gain for the PID |
| Reset | Reset | Reset constant for this PID |
| Rate | Rate | Rate constant for this PID |
| Setpoint | Setpoint value to use as the initial value. | Name of the variable acting as setpoint or a |
| Setpoint Max. Scale | SP_Max | Engineering unit value for 100% scale or blank to use 100% |
| Setpoint Min. Scale | SP_Min | Engineering unit value for 0% scale or blank to use 0% |
| Process Variable | Process_Var | Name of the process variable |
| Process Var. Max. Scale | PV_Max | Engineering unit value for 100% scale or blank to use 100% |
| Process Var. Min. Scale | PV_Min | Engineering unit value for 0% scale or blank to use 0% |
| Control Variable | Control_Var | The output of the PID loop - analog output channel or floating point variable. |
| Control Var. Max. Scale | CV_Max | Engineering unit value for 100% scale or blank to use 100% |
| Control Var. Min. Scale | CV_Min | Engineering unit value for 0% scale or blank to use 0% |
| Bias | Bias | Amount to be added to the Output |
| High Limit | High_Limit | Maximum allowable value for the Output |
| High Limit Max. Scale | HL_Max | Engineering unit value for 100% scale or blank to use 100% |
| High Limit Min. Scale | HL_Min | Engineering unit value for 0% scale or blank to use 0% |
| Low Limit | Low_Limit | Minimum allowable value for the Output. |
| Low Limit Max. Scale | LL_Max | Engineering unit value for 100% scale or blank to use 100% |
| Low Limit Min. Scale | LL_Min | Engineering unit value for 0% scale or blank to use 0% |

## Starting PID Loop Execution

The PID Loop is started in the program when the Start_PID keyword is executed during normal program sweep. For example the statement:

```
Start_PID Tank_Level.
```

starts the PID Loop named Tank_Level. Care must be taken that the Start_PID operation is executed only once by making sure it is scanned only once. Each time the Start_PID keyword is executed, the loop restarts its calculations with the starting parameter values.

To stop a PID Loop use the Stop_PID keyword. For example the Statement:

```
Stop_PID Tank_Level with 10.
```

stops the PID Loop named Tank_Level and sets the output to 10. The ending WITH followed by a value is optional.

Once the loop is executing, it continues its operation until stopped by the program. The loop operates in the background not depending on which current States are active or any other program operations.

# On-Line PID Loop Tuning

To tune a PID loop select the PID Loop option from the Debug Mode menu. This option is displayed only if there is a loop defined in the project.

The Gain, Reset, and Rate constants can be adjusted to obtain the desired PID performance, such as speed of response and over shoot. In addition, each PID has a high limit and low limit that can be set to limit the output to less than its full range when desired. These limits automatically employ anti-reset windup. In addition the Setpoint values can be adjusted from the tuning form.

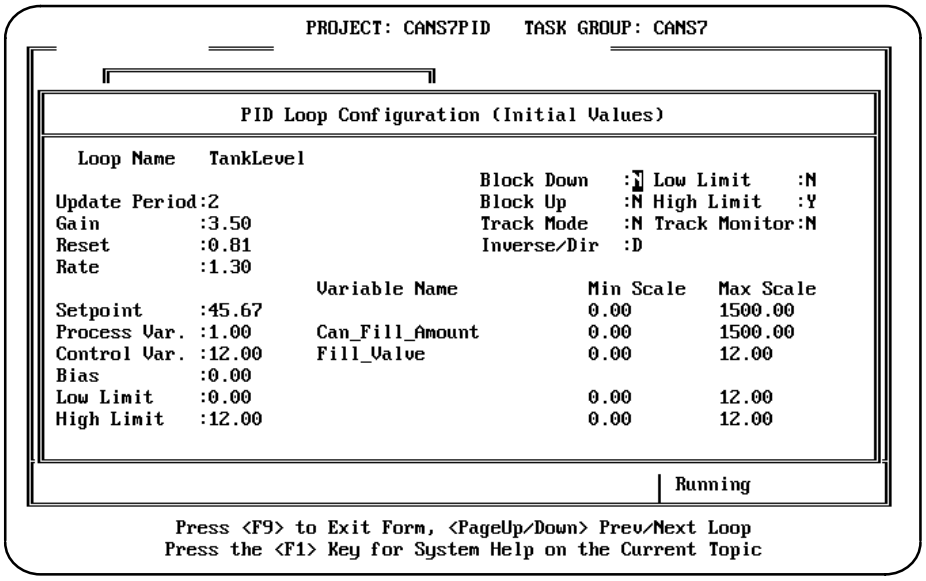
PROJECT: CANS7PID TASK GROUP: CANS7
PID Loop Configuration (Initial Values)
Loop Name TankLevel
Block Down :N Low Limit :N
Update Period:2 Block Up :N High Limit :Y
Gain :3.50 Track Mode :N Track Monitor:N
Reset :0.81 Inverse/Dir :D
Rate :1.30
Variable Name Min Scale Max Scale
Setpoint :45.67 0.00 1500.00
Process Var. :1.00 Can_Fill_Amount 0.00 1500.00
Control Var. :12.00 Fill_Valve 0.00 12.00
Bias :0.00
Low Limit :0.00 0.00 12.00
High Limit :12.00 0.00 12.00
Running
Press <F9> to Exit Form, <PageUp/Down> Prev/Next Loop
Press the <F1> Key for System Help on the Current Topic

**Figure 8-2. PID Tuning Screen**

Within the PID algorithm, all signals are treated as being 0 to 100%. Each input and the output and the high low limits, have scaling constants associated with them. Values given for this parameter are converted to a percentage of the range specified by the scaling constants.

If these constants are left blank at programming time, that input is assumed to be already 0 to 100%. Values given for this parameter are assumed to be a percentage which has already been scaled.

By using the scaling factors, the output can be scaled to have a live 0, that is go from -100% to +100%. This is a valuable tool at times when cascading PID's and the upstream PID needs the ability to overcome and move the downstream PID across its full range.

# Program Control of PID Loops

The PID loop is started in operation by the execution of statements in the program. All of the parameters that can be adjusted from the tuning form can also be changed from program statements. There are also control and status bits used in the program to either sense a condition of the loop or control its operation.

## Program Changes to the Tuning Parameters

The program refers to the loop parameters by specifying the name of the PID loop followed by the parameter keyword. For example, the ECLiPS program statement:

```
Make Tank_Level Setpoint = 45.
```

sets the setpoint of PID loop named Tank_Level to 45.

Any of the parameters including Gain, Reset, and Rate can be adjusted by the program. This capability allows for flexible control to tune the loop automatically with program instructions.

## Using the Command and Status Bits

There are also PID status and command bits associated with each PID loop. These bits indicate information about the status of the PID loop or may be used to control the PID loop. To use these bits in an ECLiPS program, the PID name is used followed by the keyword for the bit.

The ECLiPS statement:

```
If Tank_Level High_Limit_Status is true, go to Manual_Mode State.
makes Manual_Mode the active State of the Task if the output of Tank_Level
PID loop is at the high limit value.
```

The ECLiPS statement:

```
Set_Bit Tank_Level Track_Mode.
```

makes the Tank_Level PID loop tack the output so that there is no error signal.

**Table 8-2. PID Command and Status Bits**

| Status or Command Bit | Description |
|---|---|
| Block_Up | When this bit is set, the PID does not integrate up for a positive error signal. Used for Anti_Reset Windup. |
| Block_Down | When this bit is set, the PID does notintegrate down for a negative error signal. Used for Anti_Reset Windup. |
| Track_Mode | When this bit is set, the PID tracks the output so that no error is calculated. Used for a bumpless transfer to automatic mode. |
| High_Limit_Status | When this bit is set, the PID output has reached the high limit parameter or Block_Up for this PID is true. This is a read-only bit. |
| Low_Limit_Status | When this bit is set, the PID output has reached the low limit parameter value or Block_Down is true for this PID loop. This is a read-only bit. |

The Track_Mode input when set, puts the PID loop into a track mode where the output is not changed by the PID, and the algorithm is set up to perform a bumpless transfer when the Track_Mode input goes back to being false. This signal would be used with a Manual / Automatic station to obtain Manual / Auto bumpless transfer.

The other two inputs, PID_Block_Up and PID_Block_Down limit the PID loop to integrating in only one direction and tracking in the other direction. For example, when Block_Up is true, the PID is allowed to integrate down, towards zero, when the error signal, the difference between setpoint and process variable, is negative. If the error signal is positive however, the output will not change and the integral portion will not be allowed to wind up. Block_Down works exactly opposite.

These two inputs can be used to provide Anti-Reset Windup when one PID is cascaded into another PID. If during the course of operation the down stream PID reaches the point it can no longer integrate in response to the upstream PID's output, as when it reaches full scale, then any further integrating of the upstream PID would be counter productive. These inputs can stop that excess integration in the counter productive direction while allowing immediate response in the opposite direction which is the direction that will have an affect on the down stream PID.

Using the Track, Block_Up and Block_Down signals allow the User to build very sophisticated controls. If all that is needed is a simple single loop controller, these inputs can be ignored. A more detailed discussion of these inputs and their use is in the PID Algorithm Philosophy Section.

## PID Algorithm and Philosophy

The Series 90-70 State Engine PID employs a traditional algorithm that compares a setpoint with a process variable to generate an error signal. The error signal is acted upon by any or all of three parts; proportional, integral, or derivative, and the resulting output is the action that should be taken by the process actuator.

Each of the three parts has a tuning constant associated with it that can be adjusted to affect how the control action occurs. The Proportional part or term uses the Gain tuning constant. Its result is simply the product of the error, the difference between the process input and the setpoint, multiplied by the Gain. It is an instantaneous value that changes as the error changes.

The Integral term uses the Reset tuning constant. Its result is an accumulation of the product of the error signal times the Reset over time. Even though an error signal is currently zero value, the integral portion may provide a result because previous error signals have accumulated.

The Derivative term uses the Rate tuning constant. The derivative term's result immediately allows an error signal to have its full effect, then returns the term's value to zero as time goes on. The amount of the Derivative term output for a given error and the rate it decays is affected by the value of the Rate tuning constant.

The total output of the PID is the sum of the results of the three terms. Figure 1 shows a simplified diagram of the algorithm. Typically the Proportional and Integral terms are used more often alone without Derivative because this provides a more stable control performance. The Derivative term allows more anticipation and quick response, but at a penalty of possible over response and undesirable process disturbances.

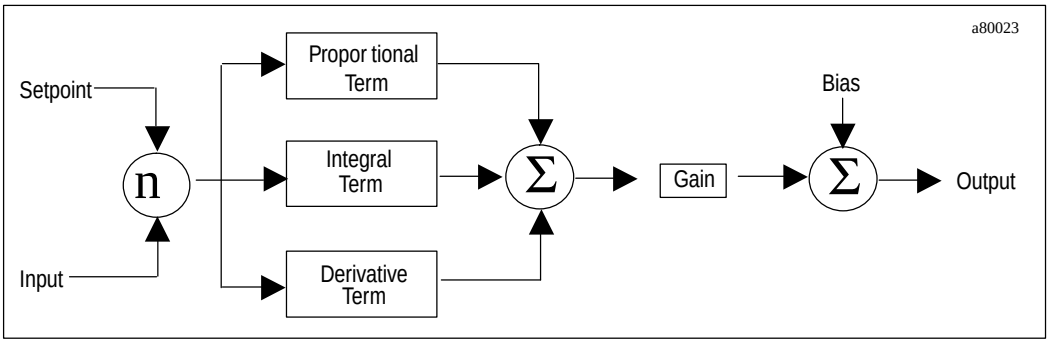


**Figure 8-3. PID Algorithms**

When the process variable differs from the setpoint, such as at the time of a step change in the setpoint, the proportional term immediately causes the output of the controller to change. As time passes the integral term integrates the controller in the same direction. The action of the controller hopefully brings the process variable closer to the setpoint. This causes the error to become smaller and decreases the proportional term, but the integral term continues to increase as it adds on the error signal over time.

Ultimately the process variable equals the setpoint and the error is zero causing the proportional term's value to be zero. In addition, the integral portion is no longer changing because the error is zero, therefore the controller output remains constant equal to the value the integral term accumulated. Any changes in process variable or setpoint cause an error and the controller will integrate to adjust the output to bring the system to equilibrium.

The addition of the derivative term, makes the controller react more extremely when the error is first detected. Then as a function of the Rate tuning constant, this reaction decays out allowing the integral term to bring the system into balance and remove the error.

## Simple PID Control

Many applications only require a simple PID loop to achieve the desired control results. A process variable represented by an analog input is compared with a setpoint with the output of the PID controller being directly sent to a field actuator by means of an analog output.

To set up this simple loop, the User needs only choose the "Define PID" function from the "Define" menu in ECLiPS. The User then enters the name of the analog input, the name of the analog output, and the various initial settings for the tuning constants.

The setpoint and tuning constants can be changed while the process is on line by using the "Tuning" function in either the Debug mode of ECLiPS or OnTOP.

## Complex PID Control

In some cases a more complex strategy using PID algorithms may be desired. Rather than the output of the PID going directly to an analog output, a cascaded PID strategy can be used in which the output of one PID goes to the input of another PID.

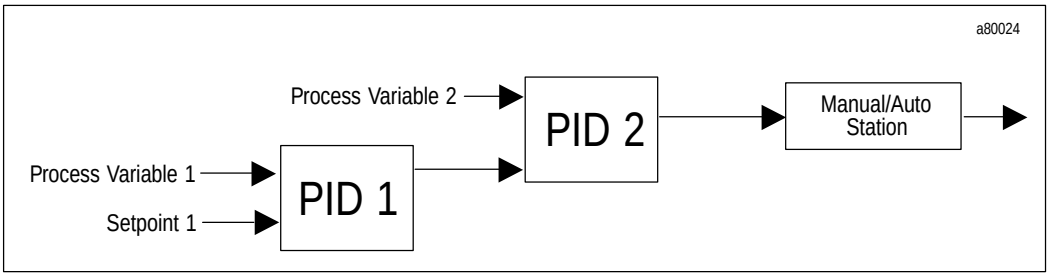


**Figure 8-4. Cascaded PID Loops**

In this example PID 1 compares process variable PV1 with the desired setpoint. The output of that controller is then fed into PID 2 as a setpoint and is compared with process variable PV2. Generally the second PID (PID 2) called the downstream PID, will be tuned to have a faster response time. It will act first to move the controller quickly in the right direction, and then the slower acting upstream PID will act to integrate out the error between the control variable and its desired setpoint.

The PID algorithm can use an analog input as its input, as in the case of PID 1 or any floating point variable. In the case of PID 2, the input to PID 2 can be defined as the output of PID 1. Likewise the setpoint can be a constant, or a named variable or as in this case, an analog input. To set this strategy just use the “PID Define” menu and name the input and setpoint as described in Section .

## Bumpless Transfer

Figure 2 also shows a Manual / Auto station between the PID output and the actual analog output card. This station allows the User to place the station in Manual, and then by means of Raise and Lower push buttons, change the actual value of the Manual / Auto output. In the Auto mode, the value of the Manual / Auto station is equal to the output of the PID.

When the M / A station is in Manual, the station value can and usually will be forced to a value other than one that will make the setpoint equal the process variable input. If the PID followed its normal operation, the integral term would continue to integrate because of the error signal between the process variable and setpoint. This would leave a difference between the PID output, which is the Auto input to the M / A station, and the actual M / A output. Then when Auto mode is selected, this difference would cause a jump or bump in the M / A output. This would upset the process and is desirable to avoid.

To prevent this bump from happening, the PID needs to have another mode of operation besides its automatic mode. In this mode, called tracking, the PID output will be maintained at what ever value it is set to, such as the M / A output in this example. The PID will not perform its normal arithmetic, but will instead set itself so that when tracking is removed, the PID output gradually goes to the proper value and avoids the bump. This is called bumpless transfer.

Each of the twenty PID algorithms have logical signals associated with them that use the name Track_Mode and the actual PID name. As an example, if the User named the first PID algorithm Tank_Level then the logical signal would be called Tank_Level Track_Mode. When Track_Mode is made true, then the PID automatically discontinues its normal algorithm, and begins tracking its output and preparing for bumpless transfer.

The User can set the Track_Mode variable from any active State. Using this variable, the User can create any tracking strategy he desires.

## Anti-Reset Windup

Using a cascaded PID strategy can cause the User some subtle problems. In the example shown in Figure 2, if PID 2 has reached its maximum and PID 1 still has an error signal because the setpoint does not equal PV1, then PID 1 would continue to integrate. The output of PID 1 would continue to increase, but it would have no affect on PID 2 since it is already at its maximum. However when the error signal of PID 1 reversed direction and caused PID 1 to begin integrating in the opposite direction, PID 1 would have to integrate below the threshold value it was when PID 2 reached its maximum before it would have any affect on PID 2. This excess amount of output PID 1 has accumulated is commonly referred to as reset windup.

The upstream controller, PID 1, needs to be prevented from winding up. An input called Block_Up will transfer the PID algorithm into a mode such that it will not integrate in the Up, towards 100%, direction. This is called anti-reset windup.

The PID will still be able to integrate down if the error signal reverses direction. That is, if the process variable is less than the setpoint the PID will not integrate up. If the process variable becomes greater than the setpoint the PID will integrate down. The Block_Down input works exactly opposite.

In the case of Figure 2, if the setpoint for PID 1 is greater than PV1, PID 1 output will continually increase. This is the setpoint for PID 2 and assume it is already greater than PV2 and PID 2 has reached its high limit. There is no profit in PID 1 output getting larger since PID 2 can not respond to its demand, PID 2 is already at its limit. Therefore the User should set the PID_1 Block_Up input true and stop the PID from winding up further.

Then when either PV2 increases or the high limit on PID 2 is changed which will allow further action by PID 2, the PID_1 Block_Up input can be set false and PID 1 can resume integrating. Or if PV1 rises above the setpoint in response to the control action, PID 1 will begin integrating the output of PID 1 in the lower direction which is permissible. When PID 1 output falls below the input PV2 into the downstream PID 2, PID 2 will integrate down below its high limit and remove PID_1 Block_Up and return the complete loop to normal.

Using the Block_Up and Block_Down inputs any amount of cascading of PID's can be accomplished without reset windup occurring and with bumpless transfers from one mode to another.

# Chapter 9 On-Line Features

This chapter describes the features that are used to work On - Line with the controller. ECLiPS controls the On - Line conditions of the controller through the use of Debug Mode. Debug Mode is accessed from the Main Menu or by pressing <F10> from the Program Mode or selecting the “Debug Mode” option from the main menu..

Debug Mode is used to observe or change the current State of a task, the value of a variable, or the condition of an I/O point. In addition the program can be viewed and downloaded, faults inspected and cleared, data logged to printer or to disk, and simulation mode status changed.

# Debug Mode Display Screen

The Debug Mode Display Screen is the main interactive link between ECLiPS and the State Engine. The top of the screen displays which project is being run in the controller. At the bottom of the screen directions for use and control keys are displayed. The screen is split into three windows.

The largest window, the Terminal Log displays any communication to or from the controller. Communication history can be accessed by pressing the up arrow key. The lower right corner of the screen displays the Controller status, Running, Halt, or No Comm Connect. It also displays Forced if any variables are being forced. The lower left corner of the screen displays any run time errors generated by the State Engine.

Several of the features used in ECLiPS temporarily replace the Terminal Log. While maintaining the same border, a different heading is displayed in the upper left corner of the Terminal Log border. The keyboard stops responds to a different set of commands listed across the bottom bar of the display. To return to the Terminal Log press <ESC>.

The Debug Mode Menu is accessed by pressing the <F3> key. All of the features and functions of the Debug mode can be controlled through the Debug Mode Menu. There are a number of Hot Keys that can be used to initiate frequently used commands directly. The help menu can be accessed by pressing the <F1> key at any time. The help is context sensitive and will provide help on the area being used.

# Controlling the Project

The Project option under the Debug Mode Menu controls the project in the controller. The Run the Program and Halt the Program options are used to control the status of the State Engine.

## Logging Data

All of the text displayed in the Terminal Log may also be logged to the hard drive of the ECLiPS computer. A hard copy can also be generated by sending the data to a printer connected to the parallel port, LPT1.

To log data from the program, use WRITE TERMs to send data to the ECLiPS Port. This data is displayed in the Terminal Log and therefore can be captured by the logging functions. When logging to a hard drive, a file is specified. If the file already exists, the file can be either overwritten or appended.

## Process Simulation

Simulation mode can be toggled on or off from the PROJECT Menu. When simulation mode is on, the State Engine does not interact with the I/O. Outputs are not sent to the output modules and inputs from the field are not seen by the program.

This function is used to test programs without connecting to real world I/O. Inputs may be controlled with the force function. A very useful way to control the inputs is writing a simulation Task that turns on the inputs just as the real world machine would. Contact Adatek for an application note on simulation.

### Download a Project

The download option can be used to download a new project to the State Engine without leaving Debug Mode. The current project in the controller must be halted before a new project can be loaded with this operation.

### Reset and Clear State Engine

The "Reset and Clear the State Engine" option on the PROJECT menu is used to clear the State Engine Program Memory Area and to set all configuration parameters to the default condition. After this operation ECLiPS displays two choices, download a project or return to Program Mode.

## Observing State Engine Values

There are four menu options to observe State Engine data from the debug mode: Monitor Tables, View, Trace, and Display Data. Monitor tables provide a real-time view of a limited number of variables. The View option displays the program allowing the value of the word at the cursor to be displayed. The trace is used to track the transitions from State to State in the State Engine. Display data is a quick way to get a snap shot of the data of a single data point.

### Monitor Tables

The monitor command is used to create, display, modify, and delete monitor tables. A monitor table is an ongoing display of current values of selected elements. The conditions of these elements are displayed below the Terminal Log. Six elements from the following list can be combined to make a monitor table:

String and Character Variables

Digital Points

Internal Flags

Analog Channels

Numeric (Integer and Floating Point) Variables

Reserved System Variables (Time, Date, Etc.)

Current State of a Task

Each Monitor table can show information for up to 6 data elements. There can be a total of 10 different monitor tables, although only one can Monitor table can be shown at a time. If the user wishes to view a different Monitor Table the Select Monitor Table option should be used.

Monitor tables can be added, modified, or deleted through the use of the commands in the Monitor Menu. The bottom of the screen will provide instructions on which keys to use to accomplish these tasks.

The Clear All Monitor Tables command can be used to clear all of the settings for all of the monitor tables. Remove the Current Monitor Table will eliminate the settings for only one monitor table.

## View

The View is a very versatile way of observing the operation of the State Engine. The View feature allows viewing of; the State Logic Program English text, system information, or the current state of all the tasks. When using the View command, the Terminal Log is replaced with the View Screen. In order to return to the Terminal Log the user must press <ESC>.

The View English text option displays the State Logic program text while still in Debug Mode. Features in the View English Text mode are accessed through the <Alt + F> Key. The search feature looks for particular words or tasks. Placing the cursor on a variable name will give the value of that variable. If the cursor is located on a task name, the current state of that task is displayed and can be changed.

Variable and Digital I/O values can also be displayed and changed from the View English Text Mode. Place the cursor on the desired word and use the <Alt + F> option.

The View Event Queue command will give the user information about the controller, such as: Run, halt, power up, run time errors, etc. The information is time and date stamped. The version of the State Engine is also displayed here.

System Status information, Scan rate, and memory usage can be shown using View System Status. The scan rate information specifies the average scan rate over the last 1000 scans. If the scan time is displayed as 0ms then the State Engine has not had enough time to complete a thousand scans. Wait awhile and try again.

The current state of every task can be shown using the View command. This display contains real-time information so the State information is updated as the data changes. This option is a very powerful troubleshooting and debugging tool. The current States provide important information about what the process is currently doing.

## Trace

The Trace is a history of the most recent program State transitions. This option allows the display of the history of State transitions on the screen for all tasks or selected tasks.

The Trace display is stored in a file that may be printed by using the Program Mode "Print Project Data" option from the PROJECT menu. The transitions are displayed sequentially, most recent items at the top. The display shows that Task name, the starting and destination State and the time that the transition occurred. The cursor can be moved down through the trace or use the <ALT + F> key to search for information.

The Trace can be customized to trace the transitions in specified Tasks only. By limiting the Tasks, transitions in irrelevant Tasks can be prevented from filling up the Trace display.

## Display Data

ECLiPS allows the user to observe the value or condition of any variable or task through the Display command. When using the Display command the user selects the data type and the variable he wishes to observe and the value or condition will be displayed to the Terminal Log.

## Faults

ECLiPS allows the user to observe and clear the PLC fault tables. The PLC fault table will display information such as "module hardware fault". The I/O fault table will show any faults in the PLC I/O system.

# Changing State Engine Values

There are three functions used to change data values in the State Engine. The View English Text can be used to change data values, and is covered under the Observing State Engine Values section.

## Change Variable Data

The Change Data option can be used to change the value of string, character, and numeric variables. The Change command can also be used to change the current State of any Task. When the change command is used a message is sent to the Terminal Log, showing what change was made.

## Force Inputs and Outputs

This feature allows the user to force digital I/O, Internal Flags, and Analog Channels to desired conditions. When forced, inputs ignore real world sensor signals and outputs ignore program instructions. To change a forced condition, use the force option to change the data or to remove the forces. After the force condition is removed, the input values returns to the real world status and the outputs are again controlled by the State Logic program.

# On-Line Hints

The structure of State Logic programs makes the key to trouble shooting and debugging programs is to concentrate on the current States of the process. By knowing the current States, you know what part of the program is executing and therefore which outputs are ON, and what conditions must be satisfied to move to the next stage of the process.

When the program is running a powerful tool for watching the process is to have the current State of every Task displayed on the screen. This display is available from the VIEW menu by choosing the “View Current State of Each Task”. This screen provides a view of the status of the entire process.

To troubleshoot dynamic problems use the Trace function . The Trace shows a history of State changes including a time stamp for each change. This function shows timing comparisons between Tasks and actual State changes even though the changes happen too fast to physically see the change.

When the system has stopped for some unknown reason, check the current State of the Tasks to zoom into the section of the program that is responsible for current activities.

# Chapter 10

# Serial Communications Module

Ths chater describes the Serial Communications Module (SCM), which is a separate module that inserts into the series 90-70 backplane. This module provides serial ports to the 90-70 State Logic Control System for RS-232, RS-422 / 485 communications. These ports may also be used for CCM communications. This chapter describes the module hardware and the State Logic programming that uses these serial ports.

# Overview

The Serial Communications Modules (SCM) occupies a single slot in the Series 90-70 backplane. The SCM provides serial ports for serial communications with the Series 90-70 State Engine.

Each SCM provides two serial ports, each capable of either RS-232 or RS-422 / 485 communications. There can be up to four SCMs in the 90-70 State Engine control system providing up to eight serial ports. One of those eight ports can use the CCM protocol to communicate with a host computer running a SCADA package or some custom level 2 software.

This chapter provides infomation about how to connect, configure, and install the SCM. There is also information on programming access to the ports and how to use the CCM port.

## Serial Cables

The following drawing shows the pin assignment for the Serial Communications Module ports. Both ports use the same design.

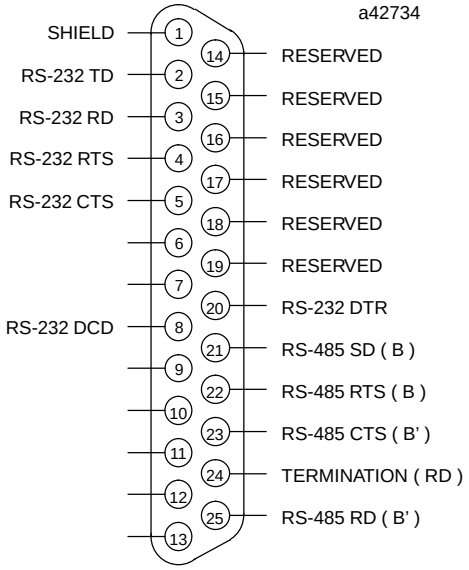


**Figure 10-1. Serial Port Assignments for Series 90-70 SCM**

NOTE: In the drawing above, the SD (Send Data) and RD (Receive Data) connections are the same as TXD and RXD used in other terminologies. (A) and (B) are the same as - and +. A' and B' denote inputs, and A and B denote outputs.

The (SCM) is designed to communicate to a device with a standard AT serial port such as most IBM PCs using the GE standard PCM to IBM-PC cable, part number IC697CBL702. If hardware flow control is not used the industrt standard null-modem cable may be used. Use the GE RS-422 / 485 serial cables if RS-422 / 485 communications are used.

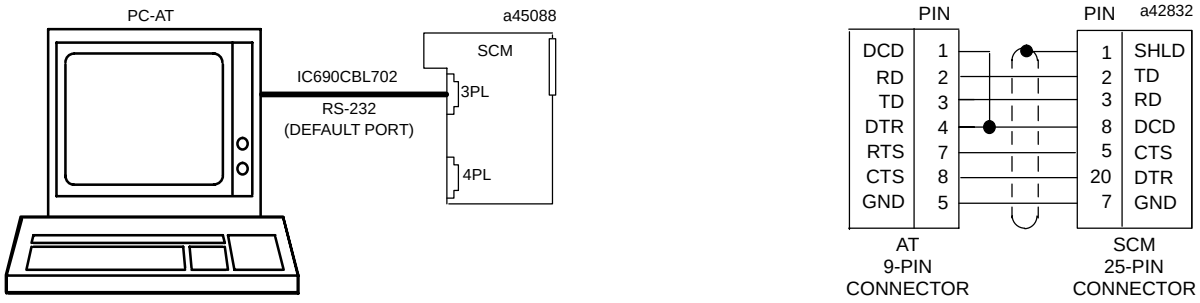


**Figure 10-2. IBM PC-AT to SCM Cable**

If connecting the SCM to Workmaster II or PS/2, use the GE PCM to Workmaster cable, part number IC697CBL705.

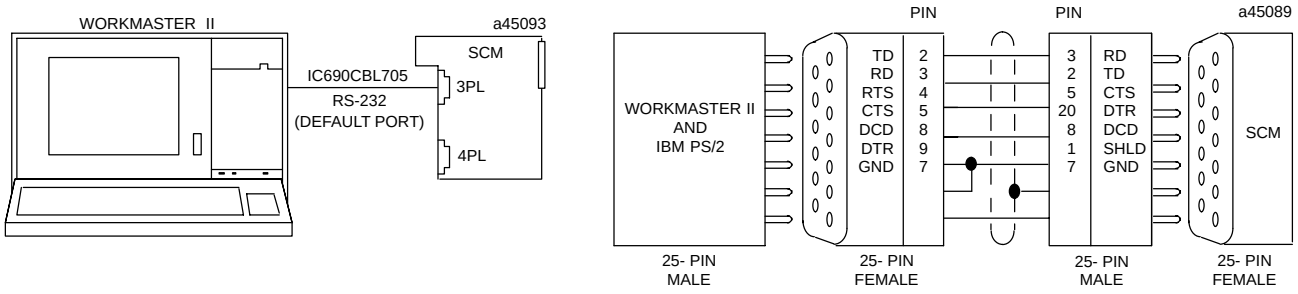


**Figure 10-3. Workmaster II or PS/2 to SCM Cable**

## SCM Fundamentals

The SCM must occupy one of four specific slots in the Series 90-70 PLC rack 0. The SCM serial ports are numbered 1 - 8, with the port number being determined by the slot location of the SCM. There are two ports on each SCM with the lowest number port for each slot being the port at the top, closest to the LEDs.

**Table 10-1. Slot Number to Serial Port Number Correlation**

| Slot Number | Port Numbers |
|---|---|
| 2 | 1 & 2 |
| 3 | 3 & 4 |
| 4 | 5 & 6 |
| 5 | 7 & 8 |

The SCM must be inserted into one of four slots (2 - 5) in the Series 90-70 rack 0. The following screen capture shows four SCMs in the 90-70 chassis, as displayed by Logicmaster.

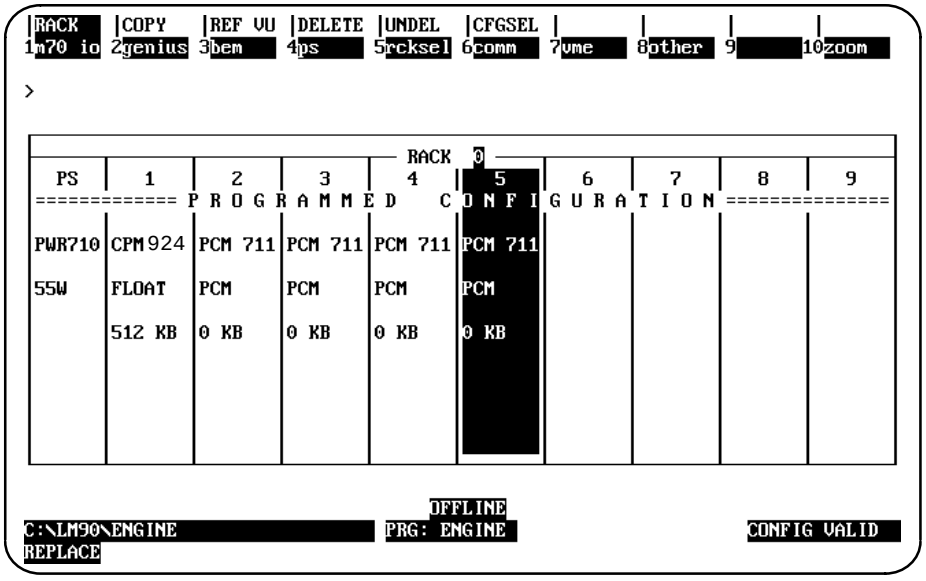


**Figure 10-4. SCMs in Series 90-70 Chasis**

For the 90-70 SCM all the slots between the SCM and 90-70 State Engine must be occupied by an intelligent module that passes interrupts. .

The State Logic program communicates through the serial ports using WRITE and READ terms. For example:

```
State: Get_Setpoint
    Write “Enter New Setpoint” to Operator_Panel.
    Read SetPoint1 from Operator_Panel, then go to Check_Setpoint State.
```

The Write Term sends the string of characters, “Enter New Setpoint”, to the port defined as the Operator_Panel. This same port is then monitored for input by the Read Term in the next Statement. The variable Setpoint1 is set to the value entered through the specified port.

In this example the WRITE and READ term information is directed to a particular port. Operator_Panel in this case is defined in ECLiPS as a specific serial port. The port name is tagged to the appropriate port (1 - 8) using the “Communication Ports” option from the LIST menu or from the “System Configuration” option of the DEFINE menu.

The Read and Write Terms may also be used without being directed to any particular channel, for example:

```
State: Choose_Recipe
    Write “Enter Recipe Number”.
    Read Recipe_Number, then go to StartBatch State.
```

Since no port is specified in this example, information is sent to the programming port using ECLiPS or OnTOP as an operator interface. For a more detailed discussion of programming serial input and output, look up the READ and WRITE terms in the Programming Instructions chapter.

# Installation and Maintenance

This section describes the SCM physical attributes. Included are a description and drawing of the module, installation and configuration instructions, and battery information.

## Description

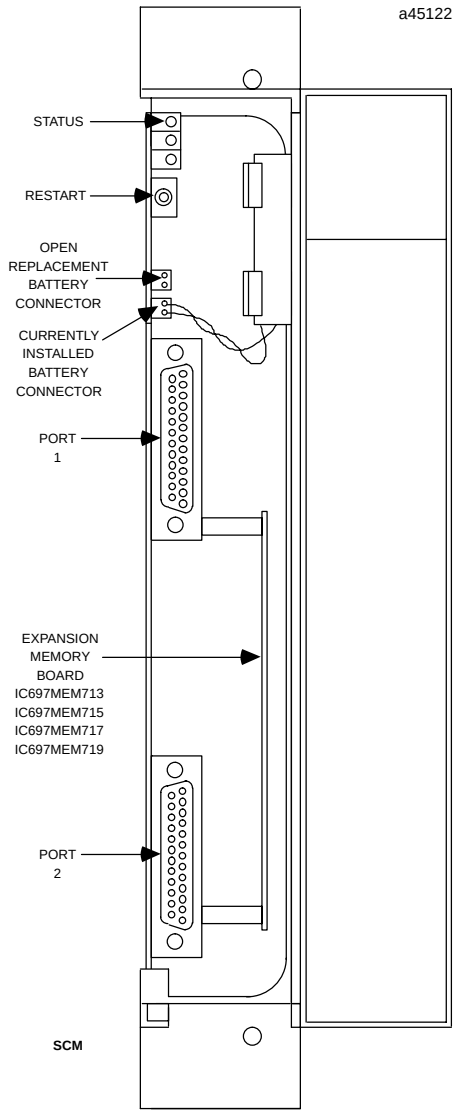


**Figure 10-5. Serial Communications Module**

There are three LED indicators located at the top front edge of the SCM. The top LED is the status indicator for the SCM and the other two are not used. During power-up this LED flashes while the SCM is running its diagnostic checks. If this LED is off, either the power is off, there is some hardware malfunction of the SCM, or there is no CPU present in the Series 90-70 PLC system. When the LED is on, the SCM is functioning normally.

The SCM comes with a battery to maintain memory when power is removed. This is a lithium battery which is installed as shown in the SCM drawings. When the battery reaches a low charge, this condition is reported to the PLC fault table. The battery circuit preserves the serial port parameter settings in memory for the SCM whenever there is no power to the module. If the battery is too low to maintain memory, the parameter settings are returned to their default state when power is restored.

Each SCM also comes with a reset switch. When the reset button is pressed for less than 5 seconds, the SCM behaves as if power were lost momentarily. Do not press the reset switch continuously for more than 5 seconds. If the reset is pressed for more than 5 seconds, power must be cycled to the SCM before it restarts correct operations.

## Installation

The SCM must be installed one of four slots, numbers 2 - 5. The slot used indicates the port number referenced in the program. There are two ports on each SCM with the lowest number port for each slot being the port at the top closest to the LEDs.

**Table 10-2. Slot Number to Serial Port Number Correlation**

| Slot Number | Port Numbers |
|---|---|
| 2 | 1 & 2 |
| 3 | 3 & 4 |
| 4 | 5 & 6 |
| 5 | 7 & 8 |

For the 90-70 SCM all the slots between the SCM and 90-70 State Engine must be occupied by an intelligent module that passes interrupts. If any slots between are empty the SCM cannot communicate with the CPU.

## Inserting the SCM

Follow these steps to insert the SCM into the Series 90 rack:

1. Power down the Series 90 PLC system
2. Locate the desired slot.
3. Slide the 90-70 SCM completely into the slot.
4. Press down firmly to lock the module in place, but do not use excessive force.
5. Power up the PLC rack. The Status LED flashes during power-up diagnostics. The LED comes on steady when the SCM is ready for operations.

## Configuration

Use the Logicmaster 90 configuration software to add the SCM to the Series 90 I/O configuration. This software is used to describe the modules present in the PLC racks. Rack and slot location and other features for each module are entered by completing setup screens that describe the modules in a rack.

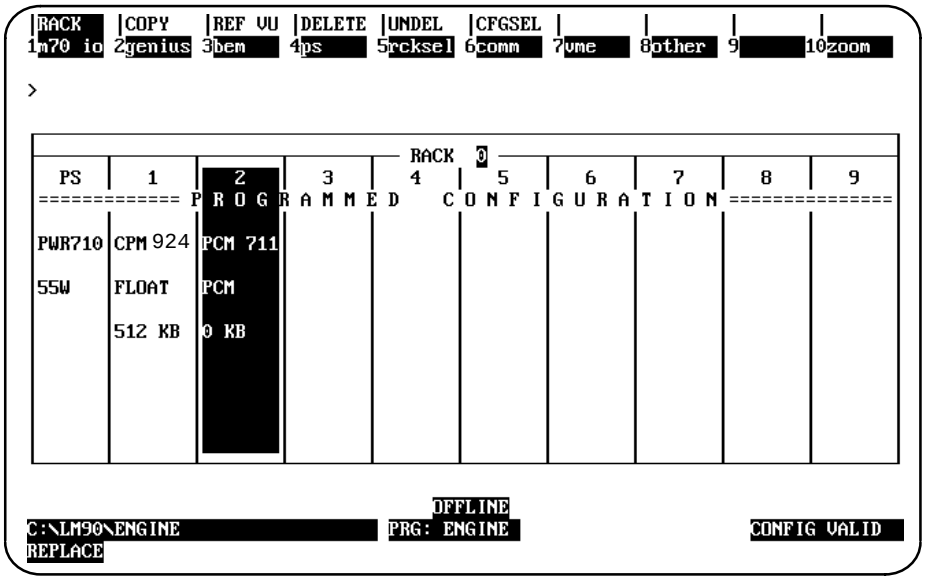


**Figure 10-6. Sample Logicmaster Configuration Screen**

From the main menu of the Logicmaster 90 configuration software, press I/O <F1>. The screen displays a representation of the modules in a rack. To add an SCM to the configuration, highlight the desired slot, then press Other <F8> and then PCM <F1>.

Now press Zoom <F10> to view the current configuration. Press <Enter> to enter the highlighted catalog number and display the PCM detail screen.

Now set the Configuration Mode to PCM CFG for the 90-70 SCM. First highlight the Config Mode option and repeatedly press the <Tab> key until PCM CFG is displayed on the screen. The serial ports are under program control and the parameters are initialized by the 90-70 State Engine each time the program is downloaded to the engine.

Now press the <Esc> key to save the configuration and return to the rack display. The display should now show a PCM in the correct slot. Send the configuration to the PLC CPU and the configuration is complete.

## Battery

The State Logic Processor comes with a 3 volt lithium battery (IC697ACC701) to maintain memory through a power cycle. If the battery charge becomes low, a fault is set in the fault table. These faults can be viewed and cleared from Debug Mode or OnTOP.

To replace the battery, connect the new battery to the extra set of battery connections then disconnect the old battery. A Product Safety Data Sheet for the battery is available. Order from GE using number GFK-0633.

# Serial Port Parameters

When first powered up the parameters for the serial ports are set to their default settings. These parameters control different aspects of how the ports interact with the serial devices connected to them.

## Parameter Details

**Table 10-3. Serial Port Parameters**

| Parameter | Settings | Description |
|---|---|---|
| Baud Rate | 19.2K - 300 | Data Transfer Rate |
| Utilize XON / XOFF Protocol | Y or N | Software Flow Control |
| Data Bits | 8, 7 or 6 | Number of bits per transmitted chara3cter |
| Parity | Even, Odd, **None** | Error checking |
| Stop Bits | 2, 1, or 0 | Number of bits indicating end of character data |
| Auto Echo | Y or N | Every character received is immediately transmitted out the transmit line |
| Receiver Always On | Y or **N** . | If the receiver is OFF any characters sent are ignored. The default setting for this parameter is for the receiver to be OFF unless a READ term is executed in the program. When the receiver is Always ON, data received when there is no active READ is saved for the next READ that is executed |
| Stop Transmit on Receive | Y or N | All transmission of charcters is stoppe when characters are being received. |
| Respond to Backspace | Y or N | When a backspace character is received the previous character received is deleted. When disabled the backspace character is merely saved as other characters would be. |
| Issue LF After CR | Y or N | A line fee is sent after every carriage return. |
| End of Message Character | Hex 0D | The ASCII value of the character that indicates that a message is (Carriage complete. When the en of message character is received the previously received characters are assigned to the READ variable and GO term following the READ is executed. |
| CCM ID #, Non Zero Enables CCM | - 90 . | CCM protocol provides for networking several devices. Each device has a unique ID number. 0 = CCM disabled |
| Enable Hardware Handshaking | Y or **N** | Hardware flow control. |
| EnableRS422 / 485 | Y or **N** | Each port may use either RS-232 or RS-422 / 485 communications standards. |

These parameter options affect the SCM serial port configuration only, not the ECLiPS or OnTOP programming port setup. The parameters for the programming port cannot be changed. The default settings are shown in bold type.

## Changing the Parameter Settings

There are two ways to change the serial port parameter settings. The first method is to modify the serial port configuration form which causes the port parameters to be set when the project is first executed after a download. The second method is to change the ports from the State Logic program function Set_Commport.

### Filling in the Port Configuration Form

To change the serial port parameter settings, fill in the configuration form. Access this form through either the DEFINE or the LIST options on the Program Mode menu. From the LIST menu select the “Communication Ports” option. Highlight the desired port from the list and and press the right arrow key to display the serial parameter form.

Access to the parameter setup form through the DEFINE menu requires that the Logicmaster configuration be imported into ECLiPS by first using the “Retrieve Logicmaster Configuration Data” option. Next use the “System Configuration” option to view the configuration dat. Highlight the slot of the SCM module to be configured and press <Enter> to bring up a menu of options. Select the “Configure Comm Ports” option.

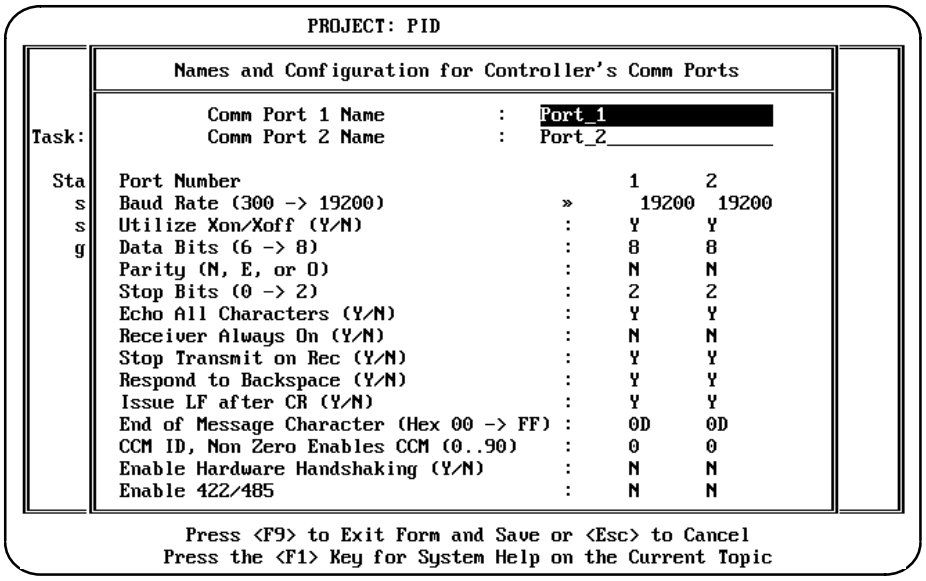
PROJECT: PID
Names and Configuration for Controller's Comm Ports
Comm Port 1 Name : Port_1
Comm Port 2 Name : Port_2
Task:
Sta
s
s
g
Port Number 1 2
Baud Rate (300 -> 19200) » 19200 19200
Utilize Xon/Xoff (Y/N) : Y Y
Data Bits (6 -> 8) : 8 8
Parity (N, E, or O) : N N
Stop Bits (0 -> 2) : 2 2
Echo All Characters (Y/N) : Y Y
Receiver Always On (Y/N) : N N
Stop Transmit on Rec (Y/N) : Y Y
Respond to Backspace (Y/N) : Y Y
Issue LF after CR (Y/N) : Y Y
End of Message Character (Hex 00 -> FF) : 0D 0D
CCM ID, Non Zero Enables CCM (0..90) : 0 0
Enable Hardware Handshaking (Y/N) : N N
Enable 422/485 : N N
Press <F9> to Exit Form and Save or <Esc> to Cancel
Press the <F1> Key for System Help on the Current Topic

**Figure 10-7. Sample Port Configuration Screen**

The form displays options for two ports at a time. After modifying the parameters press <F9> to save the settings. These settings are sent to the State Engine when the project is downloaded. The State Engine sends the configuration information to the SCMs the first time that the program is starte running after a download.

The parameters are saved over a power cycle since the SCM RAM memory is maintained by the SCM battery. If there is a problem with the battery and the parameters are lost,

they will be reset to their default settings. The parameters may be reset again by downloading the project again. The State Engine sends new serial port configuration the first time the project is run after a download.

### Program Changes to the Parameters

The serial port parameters may be controlled from the State Logic program. To change these settings from the program use the Set_Commport function.

ECLiPS provies an easy way to enter the Set_Commport function into the program. First set up the parameter options using the forms discussed in the previous section. Now place the cursor where the function is to be entered in the program an select the "Communication Ports" option from the LIST menu. Highlight the port to be changed and press <Enter>. Select the "Insert Reconfiguration Data for Port" option and press <Enter>.

The Set_Commport function call is automatically entered into the program that changes the parameters to the settings chosen in the parameter form. The parameters may be changed as often as necessary by the program. Once the changes are made, the settings remain set even over a power cycle.

## CCM2 Protocol Serial Port

CCM2 protocol is a standard open communications protocol defined by GE and documented in their manuals. The State Engine will act as a slave in a master-to-slave architecture and responds to valid CCM2 messages. The remote master computer must poll to retrieve data from or to enter data into the State Engine. The CCM2 protocol defines the message structure, framing, error checking and handling, and timing for all message types.

### Enabling CCM2 Communication

Any of the eight possible Serial Communications Module (SCM) ports can be designated the CCM2 Communications port. Select the CCM port designation from the SCM configuration options. These options are available using the "Communications Ports" option from the LIST menu or from the PLC rack display shown in the "System Configuration" option from the DEFINE menu. Select only one of the eight ports to be the CCM port.

The State Engine under CCM2 will only recognize and respond to messages sent to its address, which may be any number from 1 to 89. The default address is 0 which disables use of the CCM protocol. Any non-zero value enables CCM protocol for that port. The Station Address should be assigned to match the number the Remote Master expects and should be validated or assigned before CCM2 protocol communications is enabled.

### CCM Data Types

The CCM2, as well as many other standard protocols, were designed for communication with PLCs. To access State Engine data, the CCM host computer must identify each data element with a CCM number and a CCM type. The types of information accessible from the State Engine plus the respective CCM information are listed in the following table.

**Table 10-4. State Engine Data Types and CCM References**

| Data Type | CCM Type | CCM Number |
|---|---|---|
| Task Current State | 1 | 9501 - 9756 |
| Analog Input | 1 | 1 - 1024 |
| Analog Output | 1 | 1025 - 2048 |
| PID Loop Parameters | 1 | 4001 - 5000 |
| Integer Variables | 1 | 6001 - 7000 |
| Floating Point Variables | 1 | 7001 - 8000 |
| String Variables | 1 | 8001 - 8100 |
| Character Variables | 1 | 9001 - 9064 |
| DiscreteI / O | 2 | 1 - 45312 |
| Digital Input | 2 | 1 - 12288 |
| Digital Output | 2 | 12889 - 24576 |
| Internal Flag | 2 | 24577 - 36864 |
| % T | 2 | 36865 - 37120 |
| Global Digitals %G, %GA, etc | 2 | 37121 - 44800 |
| System Digital %S, %SA, %SB, %SC | 2 | - |

The State Engine assigns this CCM information to each data element in the program. To access this information use the “Print” option of the Program Mode Project menu. The CCM information about each data element of the State Logic program is displayed by the printout. The following is a sample of the CCM information print out:

**Table 10-5. Digital Point Names**

| Name | CCM type | CCM type # | CCM Number |
|---|---|---|---|
| Forward_Limit_Switch | Discrete Input | 2 | 36 |
| Pump_A_Starter | Discrete Output | 2 | 12289 |

**Table 10-6. Internal Flag Names**

| Name | CCM type | CCM type # | CCM Number |
|---|---|---|---|
| Part_ID_Read | Discrete | 2 | 24577 |

**Table 10-7. Analog Channel Names**

| Name | CCM type | CCM type # | CCM Number |
|---|---|---|---|
| Lube_Oil_Pressure | Register | 1 | 65 |
| Oven_Temperature | Register | 1 | 121 |

**Table 10-8. String Variable Names**

| Name | CCM type | CCM type # | CCM Number |
|---|---|---|---|
| Bar_Code_1 | Register | 1 | 8001 |

**Table 10-9. Integer Variable Names**

| Name | CCM type | CCM type # | CCM Number |
|---|---|---|---|
| Finished_Parts | Register | 1 | 6001 |
| Part_ID | Register | 1 | 6002 |

**Table 10-10. Floating Point Variable Names**

| Name | CCM type | CCM type # | CCM Number |
|---|---|---|---|
| Oven_Baking_Time | Register | 1 | 7001 |

**Table 10-11. PID Loop Names**

| Name | CCM type | CCM type # | CCM Number |
|---|---|---|---|
| Tank_Level | Register | 1 | 4001 |

**Table 10-12. Task and State Names**

| Task Name | CCM type | CCM type # | CCM Number |
|---|---|---|---|
| Start_Sequence | Register | 1 | 9501 |

| State Name | State Number |
|---|---|
| PowerUp | 1 |
| Check_Conditions | 2 |
| Start_Pump | 3 |

Each message from the Master of the CCM network specifies the type, starting register and total number of bytes to be read or written. The Master needs to specify the correct number of bytes depending upon the type of variable. The byte count and other considerations for each of the data types follow:

**Task Current State** The value stored in the designated register represents the current State of this Task. Current States can be changed and monitored using the CCM protocol. This value uses 1 register or 2 bytes in a CCM message. The inactive State is represented by a value of 0. Warning: If an invalid number is written to put a Task into a State that does not exist, the program continues to execute but that Task does nothing.

**Integer Variables** This data type uses 1 register or 2 bytes in a CCM message

**Floating Point Variables** These values use 2 registers or 4 bytes of space and are stored in IEEE format. To use these values the host computer must be able to interpret the 4 bytes as an IEEE floating point number.

**Digital Inputs/Outputs** These points are represented by the right most bit of the transferred byte. A 1 represents ON and a 0 represents OFF. Note that since digital points are OFF by default, an output set ON with CCM will only be ON for one scan and then State Engine operating system turns it OFF.

**Internal Flags** Internal flags are now a sub set of the Discrete Output. They will function like the Digital Inputs / Outputs

**Analog Inputs/Outputs** Analog I/O are floating point values and therefore require 2 registers or 4 bytes. The host must be able to interpret this value as an IEEE floating point number. Analog channels designated high speed channels are integer information using only 2 of the 4 bytes.

**String Variables** String variables can be up to 80 bytes or 40 registers long.

**Character Variables** Characters are 1 byte.

PID Loop Parameters There are 21 parameters associated with each PID loop. The register number displayed in the printout represents the number of the first parameter for the loop. The loop CCM numbers will be in multiples of 21 so that if the first loop is 4001 the second is 4022. The order of the parameters is listed in the following table.

**Table 10-13. PID Parameter Table**

| Parameter | Parameter Number | Byte Count |
|---|---|---|
| Status | 1 | 2 |
| Update Time | 2 | 4 |
| Gain | 3 | 4 |
| Reset | 4 | 4 |
| Rate | 5 | 4 |
| Setpoint | 6 | 4 |
| Setpoint Minimum | 7 | 4 |
| Setpoint Maximum | 8 | 4 |
| Process Variable (PV) | 9 | 4 |
| PV Minimum | 10 | 4 |
| PV Maximum | 11 | 4 |
| Control Variable (CV) | 12 | 4 |
| CV Minimum | 13 | 4 |
| CV Maximum | 14 | 4 |
| Bias | 15 | 4 |
| Lower Limit (LL) | 16 | 4 |
| LL Minimum | 17 | 4 |
| LL Maximum | 18 | 4 |
| High Limit (HL) | 19 | 4 |
| HL Minimum | 20 | 4 |
| HL Maximum | 21 | 4 |

The Status parameter is the only one that is not a floating point value represented in IEEE format. This parameter has information about the PID loop operation contained in the lowest 4 bits of the value of this register as follows:

Bit 0 - Block Down

Bit 1 - Block Up

Bit 2 - Track Mode

Bit 3 - Inverse Mode

If the bits are set the corresponding status is true. The other bits are reserved and should not be changed.

If the minimum and maximum values of a parameter equal 0, the parameter is considered to be a percentage in the range of 0.0% to 100.0%. Otherwise, the minimum and maximum values are used with the parameter value to calculate of percentage used in the PID calculations.

# Troubleshooting

This chapter provides procedures for diagnosing Serial Communications Module (SCM) problems. If these procedures do not solve the problem, contact the GE Hotline 1-800-828-5747 or the Adatek Hotline at 1-800-323-3343 for assistance.

## Status LED is not ON Steady

1. Check that power is supplied to the I/O rack housing the SCM. Try removing and reinstalling the SCM.
2. Cycle power to the SCM.
3. Turn the power OFF and disconnect the battery and short the SCM battery terminal connection points to clear the SCM. Reconnect the battery, turn ON power again, and reset the SCM.
4. Check that the CPU is functioning properly by checking its "OK" LED.
5. Check that there are no empty slots or non-itelligent modules that do not pass interrupts between the CPU and the SCM. If there are empty slots, the Status LED blinks continuously.
6. If the Status LED is still not ON, try to download a program from ECLiPS or OnTOP. If you can connect with the SCM and download a program, then the Status LED is faulty.
7. If you get a message that ECLiPS or OnTOP cannot connect to the controller, then check the fault table in the CPU using Logicmaster 90. If there is a fault "Bad or missing module", then the SCM is faulty and must be returned for repairs.
8. If there is no fault then contact the GE Hotline for assistance.

## Resets Blinks Port 1 or Port LED

If the Port 1 and Port 2 LEDs have an alternating blinking pattern, then the Logicmaster 90 or CPU firmware is out of date with the SCM firmware. Get updated versions of the Logicmaster software and/or the CPU firmware by calling the GE Hotline.

## Serial Communication Problems

1. Check that the serial cable used conforms to one of the types specified for communications to the SCM ports. Check that the cable is firmly secured at both ends.
2. Make sure that the serial port parameters of the serial device match the settings of the SCM. The default settings of the SCM are 19200 baud rate, 8 data bits, 2 stop bits, and no parity.
3. Make sure the RS-422 / 485 RS-232 SCM option matches the device setup.
4. If there are still problems contact the GE Hotline for assistance.

**Table 10-14. SCM Specifications**

| | |
|---|---|
| Serial Ports | 2 RS-232 / 422 / 485 |
| Memory Backup Battery | 3 Volt Lithium |
| Battery Shelf Life | 10 years |
| Battery Memory Retention with Power OFF | 6 months nominal |
| Operating Temperature | 0 to 60°C |
| Storage Temperature | -40 to 85°C |
| Humidity (non-condensing) | 5-95% |
| Vibration 1.0 G 9-150 Hz | 3.5 mm, 5-9 Hz: |
| Shock | 15 G's 11 msec |

**Table 10-15. Standards**

| | |
|---|---|
| IEC | 485, 380 |
| JIS JIS C 0911 | C 0912, |
| DIN | 435, 380 |
| UL | 508, 1012 |
| CSA C22.2 | C22.2 No. 142, |
| NEMA / ICS | 2-230.40 |
| ANSI / IEE | C-37.90A-1978 |
| VDE | 805, 806, 871-877 |
| FCC | 15J Part A |
| VME Standard C.1 | Supports VME |

# Appendix A Key Functions

This appendix lists how the keys are used by the ECLiPS and OnTOP software.

## Function Keys

| Key | Functions | | | |
|---|---|---|---|---|
| | <Key> | <Alt + Key> | <Ctrl + Key> | <Shift + Key > |
| <F1> | Help Overtype Mode | Toggle Insert Help | Project Error | |
| <F2> Disk | Save File to Controller | Send to Disk | Retrieve from | Check for Errors |
| <F3> | Menu | | | |
| <F4>Cur rent Word | Define / View Undefined Words | Define All | | |
| <F5> Replace Next | Find/ | Find Text | Replace Text | Replace All |
| <F6> | Add State | Add Task | | |
| <F7> Task | Go to Another Error | Last Project | | |
| <F8> | Mark a Block | Copy a Block | Move Block | Remove a Block |
| <F9> Form | Save / ExitForm | Next Page in | | |
| <F10> Program / Debug Modes | Toggle | | | |

## Hot Keys

| Key | Function | Mode |
|---|---|---|
| <Ctrl + D> | List Digital Points | Program / Debug |
| <Ctrl + A> | List Analog Channels | Program / Debug |
| <Ctrl + N> | List Numeric Variables | Program / Debug |
| <Ctrl + S> | List String / Character Variables | Program / Debug |
| <Ctrl + Q> | Quit Current Mode | Program / Debug |
| <Ctrl + K> | List Keywords | Program |
| <Ctrl + W> | List Filler Words | Program |
| <Ctrl + P> | List PID Loops | Program |
| <Ctrl + U> | Undelete the last Deleted Block. | Program |
| <Ctrl + R> | Communication Port Reset in Debugger | Debug |
| <Ctrl + V> | View English in Debugger | Debug |
| <Ctrl + E> | Enable Monitor Display in Debugger | Debug |
| <Ctrl + F> | Force Table | Debug |
| <Ctrl + T> | Trace Upload / Display | Debug |

## Miscellaneous Keys

| Key | Functions | |
|---|---|---|
| | <Key> | <Ctrl + key> |
| <Insert> | Add / Paste | |
| <Delete> | Remove / Cut Character / Item | |
| <Home> | Left side of Screen / Field | |
| <End> | Right side of Screen / Field | |
| <Pg Up> | Scroll Up | Top of Project / List / Menu |
| <Pg Down> | Scroll Down | End of Project / List / Menu |
| <Up> | Up 1 Line / Field | |
| <Down> | Down 1 Line / Field | |
| <Left> | Left 1 Character / Field | |
| <Right> | Right 1 Character / Field | |
| <Tab> in List / Menu | Insert 4 spaces / Next Item | |
| <Enter> Item | Carriage Return / Select | |
| <Esc> | Cancel Operation | |

# Appendix B Language Structure Summary

## Notational Conventions

This section rigorously defines the syntax for the State Logic language with a notational convention similar to that use for Bakus-Naur representations for programming languages. First the notation conventions are explained, then the syntax is represented in a top down manner starting with the program.

| | |
|---|---|
| Underline | - Identifies Keywords |
| [ ] | - Encloses terms which are optional |
| { } | - Encloses terms which may be repeated |
| < > | - Encloses a generic description of a term |
| \| | - Indicates that the term before or after may be used at this point. |
| ( ) | - Group Terms Together |

## Language Structure Notational Conventions

### Program Hierarchy

| Term | Structure |
|---|---|
| Program | { <Task Group> } |
| Task Group | { <Task> } |
| Task | Task: <Task Name> {<State>} [Start_In_Last_State] |
| State | State: <State Name> [ { <Statement> } ] |
| Statement | ([<Conditional Expression>] <Functional Expression> ) \| (<F unctional Expression> [<Conditional Expression>]) |

# Functional Structures

| Term | Structure |
|---|---|
| Functional Expression | { Functional Term } |
| Functional Term | < Turn On Discretes Term > \|<br>< Assign Values Term > \|<br>< Change Active States Term > \|<br>< Send Serial Information Term > \|<br>< PID Control Term > \|<br>< Change Serial Port Configuration Term > \|<br>< Execute Perform Functions Term > \| |
| Turn On Discrete Term | **Actuate{** <Digital I/O Name> \|<br>< Internal Flag Name > } |
| Assign Values Term | < Make Term > \|<br>< Math-Assignment Term > \|<br>< Set_Bit/Clear_Bit Term > |
| Make Term | **Make**(< Numeric Assignment Term > \|<br>< Character Assignment Term > \|<br>< String Assignment Term>) |
| Numeric Assignment Term | ( <Numeric Variable Name> \|<br><Analog I/O Name> )<br>**equal** < Numeric Value > |
| Character Assignment Term | <Character Variable Name> **equal**<br><Character Value> |
| String Assignment Term | <String Variable Name> **equal**<br><String Value> |
| Math-Assignment Term | < Add Term > \|<br>< Subtract Term > \|<br>< Multiply Term > \|<br>< Divide Term > |
| Add Term | **Add** ( < Numeric Constant > \|<br>< Variable Name> )<br>< Variable Name > |
| Subtract Term | **Subtract** ( < Numeric Constant > \|<br>< Variable Name> )<br>< Variable Name > |
| Multiply Term | **Multiply** ( < Numeric Constant > \|<br>< Variable Name> )<br>< Variable Name > |

## Functional Structures Continued

| | |
|---|---|
| **Divide Term** | **Divide** ( < Numeric Constant > \| < Variable Name> ) < Variable Name > |
| **Set_Bit/Clear_Bit Term** | **( Set_Bit \| Clear_Bit )** ( <Integer Variable Name> <Integer Number> ) |
| **Change State Term** | ( **Go** <State Name> ) \| (**Make**<Task Name>**equal**<State Name>)\| **( ( Suspend_Task \| Resume_Task )** <Task Name> |
| **Send Serial Data Term** | **Write"** <Serial Data> " [ <Port Name> ] |
| **PID Loop Control Term** | <Start PID Term> \| <Stop PID Term> |
| **Start PID Term** | **Start_PID** <PID Loop Name> |
| **Stop PID Term** | Stop_Pid<PID Loop Name> [**with** <Numeric Constant>] |
| **Port Configuration Term** | **Set_Commport** <Port Name> <Parameter Value List> |
| **Perform Function Term** | **Perform**<Function Name> **with** <Parameter Value List> |

# Conditional Structures

| Term | Structure |
|---|---|
| **Conditional Expression** | < Test Conditional > \|<br>< Character Input Conditional > |
| **Character Input** | **Read** <Variable Name><br>[ **from** <Communications Port Name> ] |
| **Test Conditional** | **If[ NOT ]** <Conditional Term><br>[{(**OR** \| **AND**)**[NOT**]<Conditional Term>}] |
| **Conditional Term** | <Digital Test Conditional> \|<br><Timer Test Conditional> \|<br><Relational Test Conditional> \|<br><Current State Test Conditional> |
| **Digital Test Conditional** | (<Digital I/O Name> \| <Flag Name>)<br>[ { (**AND** \| **OR**)<br>(<Digital I/O Name> \| <Flag Name>) } ]<br>( **ON** \| **OFF** ) |
| **Timer Conditional** | (<Numeric Constant> \|<br><Integer Variable Name>) **seconds** |
| **Current State Conditional** | <Task Name> (**equal** \| **not_equal**)<br><State Name> |
| **Relational Test Conditional** | <Numeric Relational Term> \|<br><Character Relational Term> \|<br><String Relational Term> |
| **Numeric Relational Term** | <Numeric Value><br><Relational Operator><br><Numeric Value> |
| **Character Relational Term** | <Character Value><br>(**equal** \| **not_equal**) <Character Value> |
| **String Relational Term** | <String Value><br>(**equal** \| **not_equal**) <String Value > |

# Value Expressions

| Term | Structure |
|---|---|
| **Numeric Value** | <Numeric Constant> \| <br> <Calculation> \| <br> <Numeric Variable Name> \| <br> <Analog I/O Name> \| <br> <PID Value> |
| **Calculation** | ( < Numeric Value > < Operator > <br> < Numeric Value > ) \| <br> <System Functions> (<Numeric Value>) |
| **System Functions** | SIN \| COS \| TAN \| ARCTAN \| SQRT \| EXP \| LN \| RANDOM |
| **Numeric Constant** | <Floating Point Number>\|<Integer Number> |
| **PID Value** | <PID Loop Name> <br> <PID Parameter Keyword> |
| **Character Value** | < Character Variable Name > \| <br> ' <Character> ' |
| **String Value** | <String Variable Name > \| <br> "<Character String>" - Up to 80 characters |

# Appendix C

# References to the Genius PowerTRAC Block

The PowerTRAC block is designed for power monitoring and industrial applications. The PowerTRAC block monitors voltage and current inputs and also stores digitized wave forms. From these values the block calculates voltage, current, active and reactive power, power factor, and power consumed or supplied. These values are calculated about once every second. This block is treated as an analog block by ECLiPS and the State Engine.

The current ECLiPS version supports only the PowerTRAC value calculations and the input status bits. The PowerTRAC wave form data and transient data functions are not supported at this time.

In the State Logic program each PowerTRAC block is given a name and all references to the values returned by this block are made by using the block name followed by the PowerTRAC keyword for that value. PowerTRAC keywords represent either register or status bit values.

```
If Main_Panel Voltage_A_B is less than 255, then go to the Low_Voltage State.
Write "The Accumulated Main Panel Power is %Main_Panel Accumulated_Power"
to the Status_Printer.
```

The following table lists the PowerTRAC data keywords.

| Keyword | Use |
|---|---|
| Voltage_A_B | Voltage, Phase A to B |
| Voltage_B_C | Voltage, Phase B to C |
| Voltage_C_A | Voltage, Phase C to A |
| Voltage_A_Neutral | Voltage, Phase A to Neutral |
| Voltage_B_Neutral | Voltage, Phase B to Neutral |
| Voltage_C_Neutral | Voltage, Phase C to Neutral |
| Current_Phase_A | Current, Phase A |
| Current_Phase_B | Current, Phase B |
| Current_Phase_C | Current, Phase C |
| Current_Aux | Current, Auxiliary |
| Phase_A_Power | Active Power, Phase A |
| Phase_B_Power | Active Power, Phase B |
| Phase_C_Power | Active Power, Phase C |
| Phase_A_Var | Reactive Power, Phase A |
| Phase_B_Var | Reactive Power, Phase B |
| Phase_C_Var | Reactive Power, Phase C |
| Power_Factor | Total Power Factor |
| Accumulated_Power | Total Watt / Kwatt / Mwatt–hours |

## PowerTRAC Data Keywords

There are several status bits also associated with each PowerTRAC block. The following table lists the keywords to represent the status bits set by the PowerTRAC block.

| Keyword | Use |
|---|---|
| Indicates over current on one of the Inputs | Indicates Overcurrent on one of the Inputs |
| Phase_Loop_Locked | Always 1 unless Voltage Low, Block Faulty |
| OverCurrent_On_A | Current on Phase A Over Configured Value |
| OverCurrent_On_B | Current on Phase B Over Configured Value |
| OverCurrent_On_C | Current on Phase C Over Configured Value |
| OverCurrent_On_Aux | Current on Auxiliary Over Configured Value |
| Calculation_Overflow | Value Outside Range –32768 to +32767 |

# PowerTRAC Status Bit Keywords

To reference the status bits in the State Logic program, use the same structures as are used to test other digital points.

If **Line1PowerPanel Over_Current_On_C** is on, go to OvercurrentLegC.
If **BreakerPanel Overcurrent_Captured** is on, go to Shutdown.

The current ECLiPS version supports only the PowerTRAC value calculations and the input status bits. The PowerTRAC wave form data and transient data functions are not supported at this time.

# Appendix D

# References to Genius High Speed Counter Block

The Genius High-speed Counter (HSC) Block provides direct processing of rapid pulse signals up to 200kHz. Typical applications for this block are flow meter values, velocity measurement, motion control, and material handling.

Each HSC block provides up to four counters. All program references to HSC data use a name that identifies the counter followed by one of the HSC keywords. For example:

```
If FlowMeter HSC_Strobe_Register > 456 go to PowerUp.
```

refers to the value in the strobe register of the counter named FlowMeter.

To name the counter follow these steps:

1. Retrieve the Logicmaster configuration data using the “Retrieve Logicmaster Configuration Data” option from the DEFINE menu.
2. After selecting the “System Configuration” option from the DEFINE menu, highlight the Genius Bus Controller in the 90-70 rack display and press <Enter>.
3. Select the “Configure Genius Blocks and Circuits” from the resulting menu.
4. Select the HSC block from the list of blocks and press <Enter>. Blanks are provided to name each of the four counters that can be used in an HSC block.

Some ECLiPS Statements using this reference are:

```
Make latest_count = PulseCount HSC_accumulator.
```

which saves the value of the accumulator register in the variable latest_count

```
Write “Accumulator value is %Counter_1 HSC_Counts_Per_Time”.
```

which writes the value of the accumulator to the programming port.

There are three types of data values associated with each counter: register values, status bits, and command bits. Register values refer to count values, status bits provide information about the counter, and command bits are used to control different counter

functions. Register values and status bits can only be read by the ECLiPS program. Commands bits can be both read and written to by the ECLiPS program.

The following tables lists the modifying keywords that access the counter data values. The tables list the data value referred to by the keyword along with the valid counter types for use with the keyword.

### Note

Version 1.00 of the Series 90-70 ECLiPS supports only Type A counters. If you need to use the HSC block for Type B or C features, call customer support for information on using Ladder Logic programming to make use of these features.

## Counter Register Keywords

| Register | HSC Keyword |
|---|---|
| Accumulator | HSC_ACCUMULATOR |
| Strobe Register | HSC_STROBE_REGISTER |
| Counts Per Time Base | HSC_COUNTS_PER_TIME |

## Counter Status Bit Keywords

| Status Bit | HSC Keyword |
|---|---|
| Strobe Status | HSC_STROBE_STATUS |
| Preload Status | HSC_PRELOAD_STATUS |
| Output Status | HSC_OUTPUT_STATUS |
| Module Ready | HSC_MODULE_READY |

The status bits provide information about the counter. Typical ECLiPS Statements using counter status bits are:

If **PositionCount HSC_STROBE_STATUS** is ON, go to Reset_Strobe State.
If **Encoder1 HSC_OUTPUT_STATUS** is OFF, go to Enable_Output State.

### Note

Version 1.00 of the Series 90-70 ECLiPS supports only Type A counters. If you need to use the HSC block for Type B or C features, call customer support for information on using Ladder Logic programming to make use of these features.

# Counter Command Bit Keywords

| Command Bit | HSC Keyword |
|---|---|
| Reset Strobe | HSC_RESET_STROBE |
| Reset Preload | HSC_RESET_PRELOAD |
| Enable Output | HSC_ENABLE_OUTPUT |
| Output Active | HSC_OUTPUT_ACTIVE |

The counter command bits are used to control the counter. Typical ECLiPS Statements using command bits are:

If Part_In_Place_LS is closed, turn on **Count1 HSC_ENABLE_OUTPUT**.
If Gallons_Pumped equals 987, actuate **Flow1 HSC_ENABLE_OUTPUT**.

## Note

Version 1.00 of the Series 90-70 ECLiPS supports only Type A counters. If you need to use the HSC block for Type B or C features, call customer support for information on using Ladder Logic programming to make use of these features.

# Appendix E Errors

There are two basic types of error codes. Translation error codes are generated by ECLiPS and will appear when error checking a State Logic Program. Run Time Errors are generated by the State Engine and will appear during program execution.

## Translation Errors

### Error #0

Invalid or Missing Task Name. Every Task must start with the following line: “Task: Task_Name” Task_Name may be up to 20 characters and must be a unique name in the project. It must start with a letter but may contain numbers and the “_” character. Upper and Lower case are treated equally.

### Error # 1

Invalid or Missing State Name. Every State must start with the following line: “State: State_Name” State_Name may be up to 20 characters long and must be unique within the current Task. It must start with a letter but may contain numbers and the “_” character. Upper and Lower case are treated equally.

### Error # 2

Maximum number of States reached. The maximum number of States per Task is 254. The maximum number of States per Project depends on the value in the Setup menu.

### Error # 3

Maximum number of Tasks reached. The maximum number of Tasks allowed is 255.

### Error # 4

Undefined Character in Text. The character in quotes has not been defined. Names must start with a letter and may contain only letters, numbers and the “_” character. All other characters must be defined as operators such as +, *, etc.

### Error # 5

Invalid use of the “Perform” Statement. If one of the following 3 “Perform” statements is used, it must be the only statement in the State.

Get_User_Input
User_Menu
Display_Date_&_Time

### Error # 6

Invalid use of a “Read” Statement. A “Read” may only be used along with a “Go” in a sentence and only one “Read” is allowed per State.

### Error # 7

No “PowerUp” State in Task. Every Task must have one and only one “PowerUp” State. It may appear anywhere in the Task.

### Error # 8

Duplicate Name. The name in quotes is a duplicate of another word used in the project.

### Error # 9

No other words allowed on the Task Name line. The first line of every Task must read “Task: Task_Name” and may not have any other words on it except for comments.

### Error # 10

No other words allowed on the State Name line. The words allowed on a State name line besides “State: State_Name” are the “Max_Time” definition for the State.

### Error # 11

Missing Statement. The translator was expecting to find some type of statement and found none.

### Error # 12

Invalid Command. The word or character in question is not defined as a valid command that the translator needs to start or complete the sentence.

### Error # 13

Missing Functional Term in Statement. A Conditional statement was found with no Functional statement anywhere else in the sentence. This is not allowed.

### Error # 14

Two Consecutive “ANDS” in a row.

### Error # 15

Invalid Digital or Internal Flag name. The name in the expression has not been defined as Digital Point or Internal Flag. Use the “List” or “Define” function to define it properly. All Data Elements must start with a letter but may contain numbers and the “_” character. Upper and Lower case are treated equally.

### Error # 16

Expecting Numeric Value. A numeric value is needed to complete the statement.

### Error # 17

Expecting “IN” as next word, instead of... The word “IN” or a synonym for it must appear at this point in the statement.

### Error # 18

Invalid Variable Name. The name found has not been defined as a variable. Use “List” or “Define” to properly define it. All Data Elements must start with a letter but may contain numbers and the “_” character. Upper and Lower case are treated equally.

### Error # 19

Expecting “FROM” as the next word instead of... The word “From” or a synonym for it is needed to complete the “Read” statement if a Comm Port name is used.

### Error # 20

Invalid Comm Port Name. The name found at the end of the statement must be defined as a comm port name. Use “List” or “Define” to properly define it.

### Error # 21

Expecting “SECONDS” as the next word instead of... The word “Seconds” or a synonym for it is needed to complete this type of statement.

### Error # 22

Invalid Number. The number found is out of the acceptable range of +– 1.17e–38 to 3.4e+38.

### Error # 23

Expecting quoted string instead of... A set of characters in “” is needed to complete this statement.

### Error # 24

Matching “(” not found. A “)” was found without a following “)”.

### Error # 26

Cannot use “OR” with Digitals or Flags. To obtain this functionality, place the expressions in different sentences.

### Error # 27

Missing either “ON” or “OFF” after Digital List The word found at the end of the Digital expression must be a synonym for either “On” or “Off ” in order to complete the expression.

### Error # 28

Expecting an Operator instead of.....

### Error # 29

Error Initializing Data Tables A memory error has occurred internally. If this repeats, please call Technical Support for assistance.

### Error # 30

Error Reading Project The Project File has been corrupted and cannot be read. Consult your DOS manual for instructions on how to determine the status of the file.

### Error # 31

Invalid Number or Number Variable

### Error # 32

Invalid Perform Name The Perform names are limited to those found on the List of Performs.

### Error # 34

Expecting "WITH" as next word instead of.... Either the word "With" or a synonym for it must appear in this type of statement.

### Error # 35

Missing "=" Either the "=" sign or a synonym for it must appear in this type of statement.

### Error # 36

Invalid Digital Name or Number. All Data Elements must start with a letter but may contain numbers and the "_" character. Upper and Lower case are treated equally.

### Error # 37

Number must be a 0 or 1

### Error # 38

Missing Number in Exponent The exponent operator was used without a following number.

### Error # 39

Invalid Parameter

### Error # 40

Invalid Parameter Name

### Error # 41

Expecting "Yes" or "No" instead of..... The word "Yes" or "No" must appear in this type of statement.

### Error # 42

Too many Parameters for Perform.

### Error # 43

Invalid String Constant. A string type variable may only be assigned up to 80 characters in double quotes. Any % character in a string must be immediately followed by #xx where xx represent valid hexadecimal digits, or by another % character.

### Error # 44

Invalid Character Constant. A character type variable may only be assigned a single character in single quotes.

### Error # 45

Invalid Assignment. A variable of a particular type may only be assigned values that are of the same type.

### Error # 46

Digitals and Flags Cannot be Turned OFF. The statement contains a synonym for the word "OFF" and this is not allowed. I/O that is Actuated in a State is ON, ALL others are implicitly OFF.

### Error # 47

Missing ")". A ")" was found without a preceding ")"

### Error # 48

Missing "(". A "(" was found without a preceding ")".

### Error # 49

Data Type Mismatch. An assignment or comparison was attempted between two different data types.

### Error # 50

Not Used.

### Error # 51

Two Operators used together.

### Error # 52

String Assignment is too long, Max. is 80. Because of the storage limitations in the controller, strings are limited to a length of 80 characters.

### Error # 53

Invalid or Missing Operator. A valid operator (+–=<>*/%) is needed to complete the statement in question. All "If" and "For" statements require a comparison operator (<>=) and "Make" statements require an assignment operator (=).

### Error # 54

Expecting a "." All Sentences must end with a Period.

### Error # 55

Duplicate or Missing Task Name. A State name is either missing on the line with the "State:" or the State name entered there is duplicated somewhere else in the same Task.

### Error # 56

Duplicate or Missing Task Name. A Task name is either missing on the line with the "Task:" or the Task name entered there is duplicated somewhere else in the same Project.

### Error # 57

Invalid Comparison.

### Error # 58

Misplaced Parenthesis. The "()" characters must follow operators as in "1 * (2+3)" and may not follow operands as in "1 (* 2+3)".

### Error # 59

Expression too long, it must be split. You must break the comparison into smaller multiple expressions. Try breaking it where an "And" was used or making a prior assignment for a complex mathematical expression so that only

the name must appear in the comparison.

### Error # 60

Invalid String Name. The name used has not been defined yet. You can define it with the "List" or "Define Current Word" Function. Place the cursor on the name and press the F4 key. All Data Elements must start with a letter but may contain numbers and the "_" character. Upper and Lower case are treated equally.

### Error # 61

Invalid Integer or Integer Name. The name used has not been defined yet or the number used is out of the Integer range of –32767 to 32768. You can define it with the "List" or Define Current Word" Function. Place the cursor on the name and press the F4 Key. All Data Elements must start with a letter but may contain numbers and the "_" character. Upper and Lower case are treated equally.

### Error # 62

Expecting Integer. The name used is defined as another variable type.

### Error # 63

Integer out of range. The number used is out of the integer range of –32767 to 32768.

### Error # 64

Expecting an Operand instead of.....

### Error # 65

Expecting "=" or "<>" instead of..... The operand "=" or "<>" or a synonym for one of them must appear in this type of statement.

### Error # 66

Missing Record Name of PID or Comm Port.

### Error # 67

Expecting PID Name instead of..... The name of a valid PID Loop is expected in this type of statement.

### Error # 68

Expecting "GO" after "READ" Command. A "GO" must follow a "READ" in the sentence so that the "READ" is not repeated inadvertently. For example, "Read User_Name, then go to Process_User_Name". This rule is due to the way a "READ" statement is handled by the controller and the trouble it can cause if this restriction is not imposed.

### Error # 69

Not Used

### Error # 70

Not Used

### Error # 71

Not Used

### Error # 72

String Constant too long. The maximum for a string of characters enclosed in quotes is 512. Try breaking up the statement into smaller pieces.

### Error # 73

Can't open output file. A problem has occurred accessing the disk to create the file that will get downloaded to the controller. Try the operation again.

### Error # 74

Unbalanced Parenthesis in Numerical Expression

### Error # 75

Internal Flag overflow. Too many Internal Flags have been used. ECLiPS uses them to create the code that comes from "Perform" and "For" statements. Therefore, you must reduce your use of either those types of statements or the Flags.

### Error # 76

Name too long. All names may be us to 20 characters. They must start with a letter but may contain numbers and the "_" character. Upper and Lower case are treated equally.

### Error # 77

Number out of range. The valid range for integer numbers is –32768 to 32767. The valid range for floating point numbers is +–1.1E–38 to 3.4E+38. Floating point numbers must have a decimal point followed by at least one digit. Floating point numbers may have a maximum of 7 significant digits, not counting the exponent.

### Error # 78

Power out of range, must be between –32767 and 32768. When a "^" operator is used, the following number must be an integer within the given range.

### Error # 79

Successive Power Operators not allowed. The "^" operator may not be used twice in a row.

### Error # 80

Expecting "(". The statement used must have a "(" at the proper place.

### Error # 81

Improper use of an "OR". The two statements in the sentence may not be ORed together. Only tests of numeric or character expressions may be ORed.

### Error # 82

Maximum Number of Seconds is 600.00. This is the maximum number of seconds that may be used. The resolution is 1/100 second and 60000 is the maximum value available.

### Error # 83

Mixed String and Character Types. The value of a string may not be stored in a character variable and a character variable must be assigned a single character in single quotes.

### Error # 84

Invalid use of Wait Statement. A "WAIT" may only be accompanied by a "GO" in a sentence and must appear before the "GO". This is due to the abuse this statement would endure if this restriction was not imposed. The "WAIT" statement is a simple timer that starts upon entering the State and is satisfied when the time is reached. It is intended to be used as a process delay.

### Error # 85

Expecting "GO" after "WAIT" command. A "WAIT" must be followed by a "GO" and no other statements may appear in the same sentence. This is due to the abuse this statement would endure if this restriction was not imposed. The "WAIT" statement is a simple timer that starts upon entering the State and is satisfied when the time is reached. It is intended to be used as a process delay.

### Error # 86

Digitals and Flags not allowed in "WRITE". The status of a Digital Point or Internal Flag may not be sent in a "WRITE" statement.

### Error # 87

Invalid PID Loop Name. The name in the statement is not a valid PID Loop name. Try defining it as such.

### Error # 88

Expecting Address Value. A valid address value must appear after the "@" operator.

### Error # 89

Invalid Element in Address Expression. A valid numeric expression must appear after the "@" operator.

### Error # 90

Time Elements must be set individually. Hour, Minute, and Second must be set individually as in "Hour = 13, Minute = 10".

### Error # 91

Time Value is out of range. The value of time must be between 0 and 23:59.

### Error # 92

Time Value may contain only Hours and Minutes. The time variable is accurate to only minutes, therefore it may not contain seconds.

### Error # 93

Internal String Overflow. Too many Internal Strings have been used. ECLiPS uses them to create the code that comes from the "Perform" statements for Display_Date_&_Time, and User_Menu. Therefore, you must reduce your use of either those types of statements or the Strings.

### Error # 94

Invalid PID Loop Parameter Value. An invalid value has been given to a PID Loop input or output. Use the "Define" or "List" function to enter valid values (Analog, Integer or Floating Point Variable) for all the input and output of each defined PID Loops.

### Error # 95

Not Used

### Error # 96

Invalid PID Stop Value. A PID Loop may be stopped only with a valid number.

### Error # 97

Only one PID may be started in a State. Due to the implementation of the PID Loop in the controller, only one PID may be started in a particular state.

### Error # 98

Only one "GO" allowed per sentence. Since the "GO" is always placed at the very end of a sentence, the first one will be executed and no others would ever be seen.

### Error # 99

"HALT" and "GO" not allowed in the same sentence. Since the first of the two found in the English would be executed and the other would not be seen, both are not allowed.

### Error # 100

Too Many Functional Terms in sentence. The total number of allowable characters used by Functional statements in the sentence has exceeded the maximum. The sentence should be broken up.

### Error # 101

Ambiguous Sentence, Conditionals must be Grouped. Two Conditional statements (For, If, Until, etc.) may not have a Functional statement in between them. The best way to avoid this error is to break up the sentence into smaller pieces and branch to another state if necessary.

### Error # 104

Floating Point Value Needs Leading Zero A Floating Point value must start with a number. ".1234" is not valid. You must place a leading zero on the value as follows: "0.1234".

### Error # 114

Parameter Name Mismatch. The parameter name indicated does not match the parameter name in the Perform Function Template.

### Error # 115

Missing Parameter. A required parameter is missing from the Perform Function Call.

### Error # 116

Parameter Type Mismatch. The actual parameter type does not match the type specified in the Perform Function Template.

### Error # 117

Extra Parameters. More actual parameters were found in the Perform Function Call than were specified in the Perform Function Template. A Perform statement must end in a period.

### Error # 118

Invalid PowerTRAC Monitor Parameter. The word following the name of a PowerTRAC Monitor block must be a valid parameter name. Refer to the manual for the complete list of possible parameters.

### Error # 119

Specified Block is Not Defined. The specified PCIM / BLOCK number is in the Custom I / O Scan Table, but the block is not defined. Either define the block, or remove the entry from the Custom Scan Table.

### Error # 120

Specified Block is Not Scanned. The specified PCIM / BLOCK number is defined, but has not been included in the Custom I / O Scan Table. Either delete the block, or add the entry to the Custom Scan Table.

### Error # 121

Specified Block is Not Analog. The specified PCIM / BLOCK number is defined in the Custom I / O Scan Table, and is not an Analog Block. Only analog blocks are allowed in the table. Remove the entry from the Custom Scan Table.

### Error # 122

Specified Point is Not Scanned. The specified Point is defined, but there is no entry in the Custom I / O Scan Table that will cause this point to be scanned. Either delete the point, or add an entry to the Custom Scan Table.

### Error # 123

R_Register number is out of range. The specified number is not a valid %R number.

### Error # 124

FOR expected. The keyword FOR is expected here.

### Error # 125

Invalid register length. The R_Register length is out of range. Valid lengths are 1 to 40. The length + the start register must also be within the valid range of R_Register numbers.

### Error # 126

Expecting string name. The name of a string variable is expected here.

### Error # 127

Input Capacity Exceeded. The total capacity for moving input data between the State Engine Co–Processor and the PLC has been exceeded. Remove some input variables.

### Error # 129

Mixed Inputs and Outputs. Programs written for the 9030 co–processor may not define both digital inputs and digital outputs within the same 8 bit byte.

### Error # 130

Too Many Variables of type When retentive variables and non–retentive variables of the same type are used. The total of retentive variables + non–retentive variables of that type must be at least 1 less than the system total for that type of variable. For example, the system allows 1000 integers, if 13 retentive integers are used, only 986 non–retentive integers are allowed, not 987.

### Error # 131

Expecting Colon. Every Task must start with the following syntax: "Task: Task_Name" Every State must start with the following syntax:

"State: State_Name" Blanks are allowed after the ":", but not before.

### Error # 132

Invalid Bit Field in Make Statement. Because High Speed Counter Command and Status bits are stored in a digital variables, these keywords are not allowed in the Make Statement. Reword the statement, or use the Actuate Statement to set the bit.

Invalid:

Make integer_var_1 = ctr_1 HSC_OUTPUT_STATUS.

Make ctr_1 HSC_OUTPUT_STATUS = 1.

Valid:

If ctr1 HSC_OUTPUT_STATUS is on, make integer_var_1 = 1.

Actuate ctr_1 HSC_OUTPUT_STATUS.

### Error # 133

Cannot Compare Value to Bit Status. Because High Speed Counter Command and Status bits are stored in a digital variables, an invalid comparison was made in the "IF" Statement. Reword the statement.

Invalid: If integer_var_1 = ctr_1 HSC_STROBE_STATUS, go to state2.

Valid: If integer_var_1=1 and ctr_1 HSC_STROBE_STATUS is on, go to state2.

### Error # 134

PID Bit Field in Make Statement Pid Control and Status Bits cannot be used in conjunction with the MAKE keyword. These bits must be set and cleared with the SET_BIT and CLEAR_BIT keywords.

# Runtime Errors

## Non-Critical

Integer Overflow – Value out of the normal integer range was assigned to an integer variable

Floating Point Overflow – A value outside the allowed floating point range was assiogned to a floating point variable

Divide by zero – Attempt to divide by zero

### Perform functions may generate the following errors

**BCD in called with invalid variable type:** The variable type used to store BCD data must be Float or Integer

**BCD output called with too few parameters:** The minus sign pattern and null character parameter must be defined

**Too large a number to output in BCD allocated:** There are not enough digits to show all of the numbers to the left of the decimal point

**BCD output called with invalid variable type:** The variable type used to store BCD data must be Float or Integer

**Error Define_table: called with too large a table number :** Table numbers must be from 1 to 100

**Error Define_table: table all ready defined:** Table number all ready defined the State Engine will use the first set of table definitions and ignore second definitions

**Error Define_table: called with illegal table type :** Table Type must be: I for integer, F for Float, S for string, or D for digital data type.

**Error Define_table: called with illegal save_over_halt:** Save over halt cannot be used with this data type.

**Error Copy_table_to_table: called with too large a table number:** The table number must be a value from 1 to 100.

**Error Copy_table_to_table: called with non_defined table:** The tables used must be defined

**Error Copy_table_to_table: mis_matched table types:** Both tables must be of the same type

**Error Copy_table_to_table: table to be copied larger than the other table:** The table to be copied must be smaller than the table it will be copied into.

***The Init–digital error codes are repeated for Init–float and Init–integer.***

**Init_table_digital: called with mismatched number_of_values:** The number of Number_of_values parameter must be equal to the number of values specified in the parameter table

**Init_table_digital: called with too large a table number:** The table number must be a value from 1 to 100.

**Error Init_table_digital: called with non_defined table:** The table must be defined before an Init table option can be called.

**Init_table_digital: called with non_digital table:** The table type must match the type of perform used

**Init_table_digital: called with row number out of range:** The row_number parameter must be a value that is valid for the table

**Init_table_digital called with column number out of range:** the column_number parameter must be a value that is valid for the table.

**Init_table_digital called with too many values:** The maximum no. of values that can added for each perform is 28.

***The Swap–dig error codes are repeated for Swap–float, Swap–integer, and Swap–string***

**Error Swap_table_value_dig: called with too large a table number:** The table number must be a value from 1 to 100.

**Error Swap_table_value_dig: called with non_defined table:** The table must be defined before a swap perform function is performed.

**Error Swap_table_value_dig: called with non_digital table:** The table type must match the type of perform used

**Error Swap_table_value_dig: called with row number out of range:** The row_number parameter must be a value that is valid for the table

**Error Swap_table_value_dig: called with column number out of range:** the column_number parameter must be a value that is valid for the table.

**Error Swap_table_value_int: called with illegal swap type:** The variable that is swapping the data must be of the same type as the table

**Error shift_reg par (followed by the bad parameter number):** There was an error in the way the specified parameter number was used

**Error String_manipulation starting_character parameter:** The staring_character parameter must be a value that is within the range of the string

**Error String_manipulation ending_character parameter:** The ending_character parameter must be a value that is within the range of the string.

**Error String_manipulation integer too big:** The integer value being extracted from the string is outside of the range –32767 to 32767.

**Error String_manipulation not ASCII integer value:** the value must be an ASCII integer to extract as an integer.

**Error String_manipulation not ASCII float value:** the value must be an ASCII floating point to extract as a floating point value.

**Error String_manipulation concatenated string too long:** The string to be added to the string value is too long.

**Error String_manipulation string too long:** String lengths are limited to 80 characters

**Error String_manipulation not enough digits for integer:** The number of digits being used must be equal to the number of spaces in the string

**Error String_manipulation not enough digits for float:** The number of digits being used must be equal to the number of spaces in the string

**Operation not a valid character:** When performing a String_Manipulation, the following characters are used to define the operation: s for store string, E for extract string, i for save integer, I for extract integer, f for save floating point, F for Extract floating point, C for Concatenate, L for string length, or M for match character. The operator is case dependent.

**Error Time_counter: called with wrong number of parameters:** When assigning a time counter, the time increment H for hour, M for minute, S for second, T for tenths must be specified.

**Error Time_counter: no tenth of second counters available:** The maximum number of time counters that can be used at one time is 30.

**Error Time_counter: no second counters available:** The maximum number of time counters that can be used at one time is 30.

**Error Time_counter: no minute counters available:** The maximum number of time counters that can be used at one time is 30.

**Error Time_counter: no hour counters available:** The maximum number of time counters that can be used at one time is 10.

**Error Time_counter: integer variable has already been assigned:** The integer variable being assigned has all ready been assigned to a time counter.

**Error Time_counter: integer variable has not previously been assigned:** The time counter integer variable must be assigned before being enabled or halted.

## Critical Errors

Invalid State Engine Instruction

Go to State that Does not exist

Attempt through CCM to change to an undefined State

# Appendix F Standard Predefined Key Words

ECLiPS comes with a set of Keywords supplied. Up to 10 synonyms may be added for each Keyword, and the Default Keyword may be changed. The Keywords are broken into four categories, Conditional terms, Functional terms, Operators and miscellaneous words that modify the meaning of a Statement. In the following tables, the default keyword is displayed in bold print with some suggested synonyms in normal print.

## Conditional Terms

| **Keyword,** Synonyms | Meaning / Examples |
|---|---|
| **If**, When | Test conditions and values, actions executed when test returns TRUE condition.<br>**When** the Forward_Limit_Switch is ON, go . . .<br>**If** Count is > 1, go . . .<br>**If** 3.56 seconds, go . . . |
| **Read** | Get input from comm port<br>**Read** Name from Port_1. |

# Functional Terms

| Keyword, Synonyms | Meaning / Examples |
|---|---|
| **Energize, Start, Actuate, Turn, Run, Open, Turn_On** | Turn on a Digital Point(s)<br>**Start** Conveyor_Motor.<br>**Energize** Forward_Solenoid.<br>**Actuate** Backwash_Pump, and Backwash_Pump_Light. |
| **Add** | Add a value to a variable<br>**Add** 2 to Count. |
| **Divide** | Divide a variable by a value<br>**Divide** Count by 2. |
| **Go** | Make another State the Active State<br>If Switch1 is On, **go** to the Motion State. |
| **Halt** | Stop the process immediately<br>If Alarm is On, **Halt**. |
| **Make, Put, Place, Set** | Assignment operator initiator<br>**Make** Total = 0. |
| **Multiply** | Multiply a variable by a value<br>**Multiply** Count by 2. |
| **Perform, Execute** | Invoke an ECLiPS "Perform" function<br>**Perform** Display_Date_Time with... |
| **Set_Commport** | Change comm port settings while running<br>**Set_Commport** Port_1 with Baud_Rate=9600,<br>Data_Bits=8, Parity=N,<br>Stop_Bits=2, Auto_Echo=Y,<br>Xon_Xoff=Y,<br>Receiver_On=N,<br>End_of_Message_Char=h0d. |
| **Start_PID** | Invoke a PID Loop<br>**Start_PID** Main_Loop. |
| **Stop_PID** | Halt a PID Loop and set the output value.<br>**Stop_PID** Main_Loop with 234.5. |
| **Subtract** | Subtract a value from a variable<br>**Subtract** 1 from Count. |
| **Set_Bit** | Set a condition TRUE or a bit of a 16 bit integer value to 1<br>**Set_Bit** Flowmeter_Counter HSC_OUTPUT_ENABLE.<br>**Set_Bit** Integer_Variable_1 0. |

| | |
|---|---|
| **Clear_Bit** | Set a condition FALSE or a bit of a 16 bit integer value to 0<br>**Clear_Bit** Resolver_Counter HSC_RESET_PRELOAD.<br>**Clear_Bit** Integer_Variable_2 15. |
| **Suspend_Task** | Stop the execution of a Task – no State is active.<br>If Level > alarm, **Suspend_Task** Automatic. |
| **Resume_Task** | Restart Task in State active when the Task was suspended.<br>If Level < alarm, **Resume_Task** Automatic. |
| **Write** | Send data out the comm port<br>**Write** "Error Message" to Operator_Console. |

# Operators

| Keyword, Synonyms | Meaning / Examples | Precedence |
|---|---|---|
| ( ) | Parentheses – Used to group terms to change order of of operation. Up to 18 levels of parentheses are permitted. Parentheses may be used in mathematical expressions and with relational conditional terms. | 1 |
| ARCTAN(exp) | Arctangent:<br>function returns an angle in radians where<br>–65535 <= exp <= 65535<br>Make Var = **ARCTAN**(Hyp * 2). | 2 |
| COS(exp) | COSINE:<br>exp is the angle in radians where<br>–65535 <= exp <= 65535<br>Make Near = **COS**(Test_Value). | 2 |
| EXP(exp) | e to a power:<br>Make Inverse = **EXP**(Transfer). | 2 |
| LN(exp) | Natural logarithm (base e):<br>Make Test_Value = **LN**(Input – 3.4) | 2 |
| RANDOM | Random number generator:<br>Generates a random number (0 – 1)<br>Make Sim_In = Seed * **Random** | 2 |
| SIN(exp) | Sine:<br>exp is the angle in radians<br>where –65535 <= exp <= 65535<br>Make Vector1 = **SIN**(Gauge). | 2 |
| SQRT(exp) | Square Root:<br>Make Out_Pot = **SQRT**(Flow_Meter). | 2 |
| TAN(exp) | Tangent:<br>exp is the angle in radians<br>where –65535 <= exp <= 65535<br>Make Slope = **TAN**(In_Flow). | 2 |
| ^ | Exponential Operator<br>Make Count = Amount ^ 2. | 2 |
| %, Modulus | Modulus Operator – integer operands only<br>If Count % 5 = 0, go... | 3 |
| *, Times | Multiplication Operator<br>Make Count = Amount * 2. | 3 |

## Operators Continued

| Keyword, Synonyms | Meaning / Examples | Precedence |
|---|---|---|
| /,Divided_By | Division Operator<br>Make Count = Amount / 2. | 3 |
| +, Plus | Plus Operator<br>Make Count = Amount + 2. | 4 |
| –, Minus | Minus Operator or Negative sign<br>Make Count = Amount – 2. | 4 |
| Bitwise_And | AND bits operator<br>Value = Code Bitwise_And Mask | 4 |
| Bitwise_Or | OR bits operator<br>Setup = Code Biwise_Or Mask. | 4 |
| <, Less, Under | Less than Operator<br>If Count < 5, go... | 5 |
| <=, =< | Less then or equal to Operator<br>If Count <= 5, go... | 5 |
| =, Is, Equal, Equals, Into | Equal Operator<br>If Count = 5, go... | 5 |
| >, Greater, Above, More | Greater than Operator<br>If Count > 5, go... | 5 |
| >=, => | Greater than or equal to Operator<br>If Count >= 5, go... | 5 |
| <>, Not_equal | Not Equal Operator<br>If Count <> 5, go... | 5 |
| AND | AND Operator for Conditional and Functional Terms<br>If Count > 5 **AND** Top_Switch is ON<br>Actuate Pump_5 **AND** Pump_6. | 7 |
| OR | OR Operator for Relational Conditional Terms Only<br>If Vat < 98 degrees **OR** Fuel < 12 | 8 |
| NOT | NOT Operator for Relational Conditional Terms Only<br>If **NOT** Inlet_Pressure > 100 psi | 6 |

# Miscellaneous Keywords

| Keyword, Synonyms | Meaning / Examples |
|---|---|
| State | Identifies the Name of State Logic States<br>**State:** PowerUp<br>go to the Motion **State.** |
| Task | Identifies the Name of State Logic Tasks<br>**Task:** Main<br>If the Main **Task** is in the PowerUp State, go... |
| AM | Time Suffix<br>If Time is past 3:00 **AM**, go... |
| Friday | Day of week number 5<br>If day_of_week = **Friday**, go... |
| From | Used in "Read" Terms<br>Read Name **from** Port_1. |
| Max_Time | Used to set the maximum time diagnostic for a State<br>State: PowerUp **Max_Time** 2.5 |
| Monday | Day of week number 1<br>If day_of_week = **Monday,** go... |
| Not | Logical "NOT" in a conditional expression<br>If **not** (Count > 1 or Count < 10), go... |
| Off, False,<br>Not_True,Not_Tripped | Test for Digital I/O for not set state<br>If Switch1 is **Off,** go... |
| On, True, Tripped | Test for Digital I/O for set state<br>If Switch1 is **On**, go... |
| Or | Logical "OR" in a conditional expression<br>If Count > 1 **or** Count < 10, go... |
| Inactive | Name of the State in Which No Actions Occur<br>Put the Manual Task into the **Inactive** State. |
| PM | Time suffix<br>If Time is past 3:00 **PM,** go... |
| Saturday | Day of week number 6<br>If day_of_week = **Saturday**, go... |
| Seconds | Used for comparison in a Timer Term<br>If 3.2 **seconds** have passed, go... |
| Sunday | Day of week number 7<br>If day_of_week = **Sunday**, go... |
| Thursday | Day of week number 4<br>If day_of_week = **Thursday**, go... |
| Tuesday | Day of week number 2<br>If day_of_week = **Tuesday**, go... |

| | |
|---|---|
| Wednesday | Day of week number 3<br>If day_of_week = **Wednesday**, go... |
| With | Prefix for data that is needed by a function<br>Stop_PID Kiln_Temperature **with** 45.679 |
| Start_In_Last_State | This keyword is used to configure a Task to start in the State that was active when the program stopped.<br>Task: Master_Control **Start_In_Last_State** |

# Appendix G

# Integrating Ladder Logic and/or 'C' Programming

The 90–70 State Logic Control System provides the ability to integrate ladder logic and / or 'C' programming with State Logic programming. This section explains the details of making the different systems work together.

## How the System Works

The State Engine executes in a program block that is on a rung of the 90–70 operating system. To add ladder logic or 'C' programming all that is necessary is to add additional rungs to the existing State Engine program block.

It may be necessary to use the full Logicmaster 90 version to add ladder logic and / or 'C' programming because the demo version limits the number of rungs and program blocks that can be added to the State Engine program block. To add other programming, copy the State Engine folder to another folder and add ladder logic or 'C' programming to the State Engine program block in the new folder. Use the Logicmaster 90 manuals for information on using these programming capabilities.

## Integrating the Programs

Each of the programs uses the same memory areas to store numeric or I / O information. Integrating these programs requires management of each program's memory usage, so that one program does not conflict with another program's operation.

### State Logic Outputs OFF by Default

Remember that all State Logic discrete outputs are OFF by default. The State Engine makes sure that any of the outputs defined in the program are OFF unless the program specifically turns one ON. Do not attempt to control any of the outputs defined in the State Logic program from any other program or any other device.

Forcing Outputs

Another area of potential confusion is that both ECLiPS and Logicmaster can both force discrete and analog output values. One package cannot know what forces are active from the other package. So that an output that has been forced in Logicmaster is seen to be ON in the ECLiPS package, but ECLiPS cannot see that the output has been forced.

ECLiPS can only force the outputs that have been named in the Program Mode. It is a good practice to only force State Logic outputs in ECLiPS and outputs used by other programs in Logicmaster.

Analog Scaling

Another operation that can be handled in both ECLiPS and Logicmaster is analog scaling of raw data to engineering units. To eliminate confusion and for more accurate analog representation, any analog values used in the State Logic program should be scaled only with ECLiPS.

ECLiPS uses floating point numbers to represent analog engineering unit values. Floating point representation more accurately represents the true values indicated by the analog signals produced in the field devices.

### Using State Engine Memory Location

Whenever the ladder logic of 'C' program changes memory locations, care must be taken not to conflict with State Logic operation. The different programming types should use memory locations to coordinate their activities with each other. The programs may access the same memory locations, but only one type should attempt to change a particular location.

## Register Locations

Logicmaster designates register memory with %R followed by a number and discrete memory with %M followed by a number. The State Engine uses register locations to make data available from other programming methods, the Ethernet module and other devices on the VME backplane. The %R register map as used by the State Engine are displayed in the following table:

**Table G-1. State Engine Register Usage**

| State Engine Use | Variable Bytes | Variables Allowed | Register Number |
|---|---|---|---|
| ECLiPS Communication Registers – Do NOT use | N / A | N / A | 1 - 3000 |
| Integer Variables | 2 | 1000 | 3001 - 4000 |
| Floating Point Variables | 4 | 1000 | 4001 - 6000 |
| String Variables | 82 | 100 | 6001 - 10100 |
| Character Variables | 2 | 64 | 10201 - 10265 |
| Unused | N / A | N / A | 10266 - 10900 |
| %AI and %AQ Scaled Values | 4 | 1024 of each | 10901 - 15000 |
| Current State Values | 2 | 1000 - 255 in ECLiPS | 15001 - 16000 |
| Unused | N / A | | 16001 - 16384 |

Each variable has a variable number assigned when its name is defined using ECLiPS. The variable number indicates the memory location used to store that variable's value. For example a floating point variable with number 10 would be stored in four bytes starting at %R register number 4037.

Other programs and devices may use, without fear of conflicts, any register locations not used by the State logic program. Therefore, if there were no Integer Variables used in a State Logic program, other programs and devices could use registers 3001 through 4000.

## I/O Memory Locations

The State Engine uses I/O memory space to interact with real world signals, register locations for variable and status storage, and discrete locations for internal status memory. Real world inputs can be read by all programming types, but outputs should only be controlled by one program.

The State Engine also uses discrete memory designated as %M memory in Logicmaster for internal flag values. these flags are mapped to %M locations 1 – 1000. In addition %M locations 1001 – 1500 are reserved for State Engine use and should NOT be used by other programs or devices.

There are three %M memory locations that are available as READ ONLY State Logic status information:

| |
|---|
| %M 1030 – Program Running |
| %M 1031 – Discrete Point Forced |
| %M 1032 – Analog Channel Forced |

Other discrete points that the State Logic program may control are: %Q, %G, and %T. When controlling discrete points the State Engine controls all points up to the largest number referenced in the State Logic program. The best practice is to use the lowest reference numbers possible for all points used in the State Logic program to reduce scan time and limit conflicts with other devices and programs.

For example, if the largest reference number for %G discrete points is %G500 in the State Logic program, all %Gs up to and including %G500 are controlled by the State Engine. All %Gs in that range are OFF except those turned on by the program. those not turned ON or not defined in the program are OFF. Other programs and devices should not attempt to control these points.

Analog outputs are updated in a similar fashion. all %AQs are controlled by the State Engine up to the highest %AQ referenced in the program. all unscaled outputs are updated every scan and scaled outputs are updated periodically as specified by the analog update rates specified on the tables accessed through the ECLiPS DEFINE menu. Any outputs not defined by the program but lower than the highest output referenced is set to 0 every scan. Again it is best to use to lowest %AQ reference numbers for improved scan time and fewest possible conflicts with other program and devices.

# Appendix H

# ECLiPS Specifications

| | |
|---|---|
| Tasks | 256 |
| Task Groups | 16 |
| States | 3000 |
| States Per Task | 254 |
| Statements per State | Unlimited |
| Integer Variables (range –32768 to +32767) | 1000 |
| Floating Point Variables (range +/– 1.175494E–38 to +/– 3.402823E+38) 32 BIT IEEE Format | 1000 |
| String Variables | 100 |
| String Variable Size | 80 Characters |
| Character Variables | 64 |
| PID Loops | 20 |
| Internal Flags | 1000 |
| Digital Circuits | 4096 |
| Analog Circuits | 2048 |
| Timers | Unlimited |
| Timer Resolution | 1/100 second |
| Characters per Write Term | 512 |
| State Changes Listed in Trace Display | 100 |
| Force Table Size | 32 |
| Monitor Table Size | 6 entries |
| Monitor Tables | 10 |
| I/O and Variable Names | 3000 |

# A

Actuate , 6-5
Add, 6-6 , F-2
Add Functions, 5-2
AM, F-6
Analog, 6-6 , 6-12 , 6-15
Analog Scaling, G-2
AND, 6-4 , 6-11 , 6-13 , F-5
AntiReset Windup, 8-8 , 8-11
Architecture, 2-2
ARCTAN, F-4
ASCII control characters, 6-8
ASCII variable, 6-15
Assignment Term, 6-14
Auto Echo, 10-8
AUTOEXEC.BAT, 2-3 , 2-5

# B

Battery, 2-2
Baud Rate, 10-8
Bias, 8-3
Bitwise_And, F-5
Bitwise_Or, F-5
Block_Down, 8-7 , 8-11
Block_Up, 8-7 , 8-11
Bumpless Transfer, 8-10

# C

C Programming, 2-9 , G-1
Calculations, 6-14
Cascaded PID, 8-11
CCM, 2-10 , 3-9
CCM communications, 10-1
CCM ID, 10-8
CCM Number, 10-11
CCM Type, 10-11
CCM2, 10-10
Change, 9-5
Changing Active State, 6-7
Character Variable, 4-9 , 6-15
Clear_Bit, 6-7 , F-3
Clock, 6-15
Command bit, 8-7
Comments, 4-18
Communications Port, 6-13
Conditional Expression, 6-3
Conditional Term, 3-7 , 4-7 , 6-10 , F-1
CONFIG.SYS, 2-4 , 2-5
Configuration, 2-5 , 2-6 , 2-9 , 6-15
Control Characters, 6-9
Control Theory, 3-1
Control Variable, 8-4
Copy, 5-3
COS, F-4
Current State, 3-11 , 6-7 , 6-12 , 9-5

# D

Day, 6-15
Day_of_week, 6-15
Debug Mode, 9-1
DEFINE , 5-7 , 10-9
Diagnostics, 3-4 , 3-11
Digital , 6-11
Direct acting, 8-3
Display, 9-4
Divide, 6-6 , F-2
Documentation , 4-17 , 5-5
DOS, 2-3 , 2-5
Download, 9-3
Download , 5-5

# E

End of Message Character, 10-8
Energize, F-2
Error Check, 5-5

Errors, E-1
EtherNet , 2-11
EXP, F-4
Expanded memory, 2-4
Expression, 6-2
Extended memory, 2-4

## F

Faults, 9-4
Filler Words, 4-7 , 6-4 , 6-16
FIND , 5-3
Finite State, 3-2 , 3-4
Floating Point Variable, 4-8 , 6-15 , 6-16
Force, 9-5 , G-2
Formatting, 6-8
Friday, F-6
FROM, 6-13 , F-6
Function Keys, A-1
Functional Expression, 6-3 , 6-16
Functional Term, 3-7 , 4-6 , 6-5 , F-1

## G

Gain, 8-3 , 8-8
Genius Bus Controller, 2-6
Go, 6-7 , 6-16 , F-2
Grammatical Rules, 6-16

## H

Halt, F-2
Hardware Handshaking , 10-8
Hardware Key, 2-4
Help , 1-2
Hierarchy, 6-2
High Limit, 8-3
High Speed Counter Block, D-1
High_Limit_Status, 8-7
Hints, 4-12 , 4-17 , 5-8 , 9-5
Hot Key, 5-8 , A-2
Hour, 6-15

## I

If, F-1
Inactive State, 6-7 , F-6
Installation, 2-4 , 2-7
Integer, 6-16
Integer Variable, 4-8 , 6-15
Internal Flag, 4-8
Inverse acting, 8-3

## K

Key, 1-1
Key Functions, A-1
Keywords, 4-7 , 6-4 , F-1

## L

Ladder Logic, 2-9 , G-1
Language, 6-4
Language Structure, B-1
LIST , 5-7 , 10-9
List Functions, 5-2
LN, F-4
Logging Data, 9-2
Logicmaster, 2-3 , 2-5 , 2-8 , 2-10 , 5-7 , 6-15 , 10-4 , 10-9 , G-1
Low Limit, 8-3
Low_Limit_Status, 8-7

## M

Make, F-2
Make Term, 6-5 , 6-6
Manual / Auto station, 8-10
MathAssignment , 6-6
Mathematical Calculations, 6-14
Max_Time, F-6

Memory , 2-4
Minute, 6-15
Model, 3-4
Monday, F-6
Monitor Tables, 9-3
Month, 6-15
Move, 5-3
Multiply, 6-6 , F-2

## N

Name, 4-4 , 4-17 , 5-7 , 6-4 , 6-14
Natural Logarithm, F-4
NOT, 6-13 , F-5 , F-6
Numeric Data Types, 6-16
Numeric Variable, 6-15
Numerical expressions, 6-14

## O

Off, F-6
On, F-6
On – Line Features, 9-1
Online Modify, 5-8
OnTOP, 2-3 , 2-6 , 2-10 , 6-8 , 6-14 , 8-9 , 10-5 , 10-9 , A-1
Operator Precedence, 6-14
Operators, F-1
OR, 6-4 , 6-11 , 6-13 , F-5 , F-6

## P

Parenthesis, 6-14
Password, 2-6
PASTE , 5-9
Perform, 6-5 , F-2
Perform Function, 6-10
PID Initialization, 8-2
PID Loop, 6-9 , 8-1
PID Loop Parameter, 8-3 , 10-13
PID Loop Tuning, 8-6
PID scaling constants, 8-2
PM, F-6
PowerTRAC Block, C-1
Precedence, 6-14
Print , 5-5
Process Variable, 8-4
Program Files, 2-6 , 2-8
Program Mode, 5-2
Program Scan, 4-9
Project Management, 5-4

## Q

Quick Reference, 6-5 , 6-10

## R

RANDOM, F-4
Rate, 8-3 , 8-8
READ, 6-13 , 6-16 , F-1
Read Term, 4-15 , 10-4
Receiver Always On, 10-8
Registration, 1-2
Relational Term, 6-12
Remove, 5-3
Reset , 8-3 , 8-8
Respond to Backspace, 10-8
Resume_Task, 6-7 , F-3
RS–232, 10-1
RS–422 / 485, 10-1 , 10-2 , 10-8
Run Switch, 2-2
Runtime Errors, E-13

## S

Saturday, F-6
SCADA package, 10-2
SCM Battery, 10-9 , 10-14
SCM Specifications, 10-15

Search and Replace, 5-3
Seconds, 6-11 , 6-15 , F-6
Security , 2-6 , 2-7
Serial Cable, 1-1 , 2-3 , 10-2
Serial Communications, 3-8 , 6-8 , 6-14 , 10-1
Serial Communications Module, 2-6 , 2-10, 10-1
Serial Port, 2-2 , 10-1
Serial Port Configuration, 10-9
Serial Port Parameters, 10-8
Set_Bit, 6-7 , F-2
Set_Commport, 6-10 , 10-9 , 10-10 , F-2
Setpoint, 8-4
SETUP , 2-5
Simulation, 9-1 , 9-2
SIN, F-4
Specifications, H-1
SQRT, F-4
Start_In_Last_State, F-7
Start_PID, 6-5 , 6-9 , 8-5 , F-2
State, 3-2 , 4-2 , 6-2 , 6-7 , 6-16 , F-6
State Diagram, 3-2
State Engine, 2-8 , 2-9
State Logic, 3-1
Statement, 3-5 , 4-3 , 6-2 , 6-16
Status Bits, 8-7
Stop Bits, 10-8
Stop Transmit on Receive, 10-8
Stop_PID, 6-5 , 6-9 , 8-5 , F-2
String Variable, 4-8 , 6-15
Subtract, 6-6
Subtract , F-2
Sunday, F-6
Suspend_Task, 6-7
Suspend_Task , F-3
Synonyms, 4-7 , F-1
System Configuration, 2-6 , 5-7 , 6-10 , 10-4, 10-9
System Status, 9-4
System Variable, 5-2

## T

TAN, F-4
Task, 3-3 , 3-4 , 3-6 , 3-9 , 4-2 , 4-14 , 6-2 , 6-16 , F-6
Task Group, 5-6 , 6-2
Task interaction, 3-10
Term, 6-2
Terminal Log, 9-2
Text Functions, 5-3
Thursday, F-6
Time Variable, 4-9 , 6-15
Timer, 4-16 , 6-11
Trace, 9-4
Track_Mode, 8-7 , 8-10
Translate, 5-5
Translation Errors, E-1
Tuesday, F-6
Tuning Parameters, 8-7

## U

Update Time, 8-4

## V

Variables, 4-7
Version, 2-6
View, 9-4

## W

Wednesday, F-7
When, F-1
WITH, 8-5 , F-7
Word, 6-2
Word Processing, 5-9
Write, 6-5 , 6-8 , F-3
Write Term, 4-14 , 9-2 , 10-4

## X

XON / XOFF, 10-8